



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения
Кафедра КБ-2 «Прикладные информационные технологии»

А.А. МЕРСОВ, А.М. РУСАКОВ, В.В. ФИЛАТОВ

Методическое указание
по выполнению курсовой работы (проекта)
по дисциплине «Языки программирования»

Москва – 2020

УДК

ББК

Печатается по решению редакционно-издательского совета «МИРЭА – Российский технологический университет»

Рецензенты:

Мерсов А.А., Русаков А.М., Филатов В. В.

Методические указания по выполнению курсовой работы по языкам программирования / А.А. Мерсов, А.М. Русаков, В. В. Филатов, – М.: МИРЭА – Российский технологический университет, 2020.

Методические указания предназначены для выполнения курсовой работы (проекта) по дисциплине «Языки программирования» и содержит перечень вариантов курсовой работы (проекта), требования к оформлению пояснительной записки, а также краткое изложение теоретического материала в форме пояснений к заданию на работу (проект). Для студентов, обучающихся по направлениям 10.03.01, 10.05.03, 10.05.04.

В методических указаниях изложен порядок выполнения курсовой работы по языкам программирования.

Материалы рассмотрены на заседании учебно-методической комиссии КБ-2 Протокол №1 от «28» августа 2020 г.

и одобрены на заседании кафедры КБ-2.

и. о. зав. кафедрой КБ-2

к.т.н.

/ О.В.Трубиенко /

УДК

ББК

© Мерсов А.А., Русаков А.М., Филатов В.В. 2020

© Российский технологический университет – МИРЭА, 2020

Содержание

ОБЩИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ	3
ЦЕЛЬ КУРСОВОЙ РАБОТЫ	3
<i>Комплексная оценка сформированности знаний, умений и владений</i>	4
ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ	4
<i>Общий план работы</i>	4
<i>Последовательность написания программного обеспечения.</i>	5
<i>Исходные данные для проектирования</i>	5
<i>Требования к программе:</i>	5
<i>Выходными данными проекта являются:</i>	5
<i>Отчет по курсовой работе</i>	6
КРИТЕРИИ ОПРЕДЕЛЕНИЯ ОЦЕНКИ ЗА КУРСОВУЮ РАБОТУ	7
<i>Общие критерии</i>	7
<i>Индикаторные критерии оценки</i>	8
<i>Сводная таблица критериев</i>	9
ВАРИАНТЫ ЗАДАНИЙ	11
НЕОБХОДИМЫЕ СВЕДЕНИЯ ИЗ ТЕОРИИ	18
ОСНОВНЫЕ СВЕДЕНИЯ ИЗ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ	18
<i>Массив символов</i>	18
<i>Символьные строки и функции работы со строками</i>	18
<i>Массивы указателей</i>	19
<i>Стандартные библиотечные функции</i>	20
<i>Преобразование символьных строк</i>	21
<i>Ссылки</i>	21
<i>Работа с памятью. Указатели</i>	22
<i>Динамическое выделение памяти</i>	24
<i>Ошибки, связанные с выделением памяти</i>	48
<i>Выделение памяти для матрицы</i>	49
<i>Известный размер строки</i>	49
<i>Неизвестный размер строки</i>	49
<i>Стандартные функции работы с файлами.</i>	50
<i>Функция открытия файла</i>	51
<i>Функция закрытия файла</i>	52
<i>Функция переименования файла</i>	52
<i>Функция контроля конца файла</i>	52
<i>Динамические структуры данных</i>	54
<i>Линейный список</i>	54
<i>Создание элемента списка</i>	55
<i>Добавление узла</i>	55
<i>Добавление узла в начало списка</i>	55
<i>Добавление узла после заданного</i>	55
<i>Добавление узла перед заданным</i>	56
<i>Добавление узла в конец списка</i>	56
<i>Проход по списку</i>	56
<i>Поиск узла в списке</i>	57
<i>Удаление узла</i>	57
<i>Барьеры</i>	58
<i>Двусвязный список</i>	58
<i>Операции с двусвязным списком</i>	58
<i>Добавление узла в начало списка</i>	58
<i>Добавление узла в конец списка</i>	59
<i>Добавление узла после заданного</i>	59
<i>Поиск узла в списке</i>	60
<i>Удаление узла</i>	60

ПАРАДИГМЫ ООП – ИНКАПСУЛЯЦИЯ, НАСЛЕДОВАНИЕ, ПОЛИМОРФИЗМ	60
ИСПОЛЬЗОВАНИЕ ДИНАМИЧЕСКИХ МАССИВОВ. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	61
Объявление динамического массива	61
КЛАССЫ. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ ИНИЦИАЛИЗАЦИИ SET() И ВЫВОДА РЕЗУЛЬТАТОВ PRINT(). ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	62
КЛАССЫ. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ С ИСПОЛЬЗОВАНИЕМ КОНСТРУКТОРА, ДЕСТРУКТОРА, FRIEND - ФУНКЦИИ ИНИЦИАЛИЗАЦИИ SET() И ФУНКЦИИ ВЫВОДА РЕЗУЛЬТАТОВ PRINT(). ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	63
Конструктор класса	63
Деструктор класса	64
Дружественная функция (friend)	64
КЛАСС «ДИНАМИЧЕСКАЯ СТРОКА» И ПЕРЕГРУЗКА ОПЕРАЦИЙ. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	64
Функции работы со строками	65
Функции преобразования строки S в число:	65
Функции преобразования числа V в строку S:	65
ПЕРЕГРУЗКА ОПЕРАЦИЙ	65
НАСЛЕДОВАНИЕ КЛАССОВ, МЕХАНИЗМ ВИРТУАЛЬНЫХ ФУНКЦИЙ. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	67
Ограничение на наследование	68
Виртуальная функция и механизм позднего связывания	68
ПРОГРАММИРОВАНИЕ ШАБЛОНА КЛАССОВ. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	69
МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ С ИСПОЛЬЗОВАНИЕМ АБСТРАКТНЫХ БАЗОВЫХ КЛАССОВ, ФАЙЛОВОГО ВВОДА-ВЫВОДА С ПРИМЕНЕНИЕМ ПОТОКОВ C++, ФУНКЦИЙ ОБРАБОТКИ ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	71
Абстрактные классы	71
Множественное наследование	71
ФАЙЛОВЫЕ ПОТОКИ. КЛАССЫ ФАЙЛОВЫХ ПОТОКОВ:	72
Конструкторы объектов (для классов ifstream, ofstream, fstream) и функции открытия/закрытия файлов:	73
ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ	73
НЕКОТОРЫЕ КРИПТОГРАФИЧЕСКИЕ ЭЛЕМЕНТЫ	76
ОСНОВНЫЕ ПОНЯТИЯ И ТЕРМИНЫ	76
Криптопровайдеры, инициализация и деинициализация	77
Шифрование	81
Расшифровывание	81
Криптографические ключи и выполняемые функции	81
Функции кодирования сертификатов:	82
Базовые криптографические функции:	82
Генерация ключей и обмен ключами	83

Общие указания к выполнению курсовой работы

Практические работы выполняются с использованием персональных компьютеров. Указания по технике безопасности совпадают с требованиями, предъявляемыми к пользователю ЭВМ. Другие опасные факторы отсутствуют.

Цель курсовой работы

Цель курсовой работы по дисциплине «Языки программирования» состоит в закреплении и углублении знаний и навыков, полученных при изучении дисциплины. Курсовая работа предполагает выполнение задания повышенной сложности по проектированию, разработке и тестированию программного обеспечения, а также оформлению сопутствующей документации.

В результате изучения дисциплины обучающийся должен:

Знать:

языки, системы и инструментальные средства программирования в профессиональной деятельности;

- математический аппарат, математические пакеты, программные комплексы;
- общие принципы построения и использования современных языков программирования высокого уровня;
- основные сведения о базовых структурах данных;
- общие сведения о методах проектирования, документирования, разработки, тестирования и отладки программного обеспечения;
- принципы построения языков и систем программирования;
- возможности библиотек программ и классов для решения различных задач;
- технологии программирования
- современные средства программного обеспечения для разработки программного продукта
- методы и принципы разработки программного и иного вида обеспечения специальных ИАС

Уметь:

использовать языки, системы и инструментальные средства программирования в профессиональной деятельности;

- строить алгоритм решения задачи, проводить его анализ и реализовывать в современных программных комплексах;
- работать с интегрированной средой разработки программного обеспечения;
- использовать современные средства разработки программного обеспечения на языках высокого уровня, библиотеки программ и классов для решения практических задач
- проектировать и реализовывать современный пользовательский интерфейс;
- применять современные средства программного обеспечения и вычислительной техники при разработке программного продукта, а также поиска и обработки информации.
- применять основные методы и принципы при разработке программного продукта и иного вида обеспечения специальных ИАС

Владеть:

- языками программирования, системами и инструментальными средствами программирования в профессиональной деятельности;
- навыками разработки, документирования, тестирования и отладки программ на языке программирования высокого уровня;
- основными методами разработки алгоритмов и программ;

- методами создания структур данных, используемые для представления типовых информационных объектов;
- основными задачами анализа алгоритмов;
- основными современными технологиями для осуществления поиска и обработки информации.
- методами и принципами разработки программного и иного вида обеспечения специальных ИАС

Комплексная оценка сформированности знаний, умений и владений

Обозначения		Формулировка требований к степени сформированности компетенции
Цифр.	Оценка	
1	Неуд.	Не имеет необходимых представлений о проверяемом материале
2	Удовл. или неуд. (по усмотрению преподавателя)	Знать на уровне ориентирования, представлений. Субъект учения знает основные признаки или термины изучаемого элемента содержания, их отнесенность к определенной науке, отрасли или объектам, узнает их в текстах, изображениях или схемах и знает, к каким источникам нужно обращаться для более детального его усвоения.
3	Удовл.	Знать и уметь на репродуктивном уровне. Субъект учения знает изученный элемент содержания репродуктивно: произвольно воспроизводит свои знания устно, письменно или в демонстрируемых действиях.
4	Хор.	Знать, уметь, владеть на аналитическом уровне. Зная на репродуктивном уровне, указывать на особенности и взаимосвязи изученных объектов, на их достоинства, ограничения, историю и перспективы развития и особенности для разных объектов усвоения.
5	Отл.	Знать, уметь, владеть на системном уровне. Субъект учения знает изученный элемент содержания системно, произвольно и доказательно воспроизводит свои знания устно, письменно или в демонстрируемых действиях, учитывая и указывая связи и зависимости между этим элементом и другими элементами содержания учебной дисциплины, его значимость в содержании учебной дисциплины.

Последовательность выполнения

Общий план работы

Для выполнения курсовой работы требуется умение писать программы на алгоритмическом языке для выполнения предложенного варианта.

Таким образом, последовательность выполнения курсовой работы следующая:

1. Ознакомится необходимым алгоритмом согласно варианту задания.
2. В соответствии с заданным вариантом написать программу.
3. Описать соответствующий алгоритм.
4. Написать и отладить на компьютере программу
5. Привести контрольную распечатку.
6. Оформить отчет.

Работа считается выполненной только после оформления отчета, защиты и подписи преподавателя.

Последовательность написания программного обеспечения.

- Шаг 1. Анализ исходных данных, указанных в задании
- Шаг 2. Определение данных, структур, классов, методов и функций, необходимых для выполнения работы согласно варианту.
- Шаг 3. Разработка соответствующего алгоритма решения конкретной задачи.
- Шаг 4. Реализация элементов, описанных в Шаге 2
- Шаг 5. Подготовка контрольных данных для тестирования программного обеспечения
- Шаг 6. Отладка разработанного программного обеспечения на основе контрольных данных, подготовленных на предыдущем Шаге.

Исходные данные для проектирования

1. информация о группе студентов из N человек, где запись о студенте содержит следующие данные:
 - 1) Ф.И.О. студента.
 - 2) Число, месяц, год рождения.
 - 3) Год поступления в институт.
 - 4) Факультет, кафедра.
 - 5) Группа.
 - 6) Номер зачетной книжки.
 - 7) Пол
 - 8) Названия предметов и оценки по каждому предмету в каждой сессии. (максимально 9 сессий и 10 предметов в каждом семестре, которые м.б. разные).
- P.S. Все данные должны быть форматными: даты, числа и т.д.
2. конкретное задание, вариант которого определяется 2-мя последними цифрами зачетной книжки.

Требования к программе:

- 1) Организовать корректировку всех данных, т.е. иметь возможность изменять данные внутри без изменения остальной информации.
- 2) На работу программы не должны оказывать влияние 1)неправильные данные 2)случайно нажатые клавиши и т.п. ,т.е. программа должна работать в любых ситуациях.
- 3) Программа должна быть разработана на основе методов объектно-ориентированного программирования(наследование, перегрузка функций, полиморфизм, использование конструкторов/деструкторов и т.д.), а также иметь файловую структуру.

Выходными данными проекта являются:

1. данные о студентах по всей группе в порядке ввода информация.
2. При выполнении конкретного задания необходимо выводить все данные по каждому студенту начиная от "Ф.И.О." и заканчивая "Текущим семестром" (по необходимости выводить промежуточную таблицу).

Отчет по курсовой работе

Отчет по курсовой работе оформляется в соответствии с требованиями, предъявляемыми к оформлению курсовых работ в вузе (Инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18)*, и должен содержать:

1. Титульный лист (установленная форма Приложение №1 в *).
2. Наименование и цель работы (берется из рекомендаций по выполнению КР).
3. Исходные данные варианта задания.
4. Алгоритм решения задачи (сведения из теории, необходимые данные, с указанием соответствующего типа, перечень используемых структур, классов с описанием данных и использованных методов и т.д.).
5. Листинг программы с исходными и выходными данными (исходный код должен быть приведен в удобном для чтения формате).
6. Анализ результатов (контрольная распечатка — скриншоты).
7. Выводы.

Замечание: листы отчета должны быть скреплены.

К отчету также должен быть приложен цифровой носитель информации (диск или флэшка) со всеми необходимыми материалами для защиты курсовой работы.

Критерии определения оценки за курсовую работу

Общие критерии

Шкала 1. Оценка сформированности отдельных элементов компетенций

Обозначения		Формулировка требований к степени сформированности компетенции		
Цифр.	Оценка	Знать	Уметь	Владеть
1	Неуд.	Отсутствие знаний	Отсутствие умений	Отсутствие навыков
2	Неуд.	Фрагментарные знания	Частично освоенное умение	Фрагментарное применение
3	Удовл.	Общие, но не структурированные знания	В целом успешное, но не систематически осуществляемое умение	В целом успешное, но не систематическое применение
4	Хор.	Сформированные, но содержащие отдельные пробелы знания	В целом успешное, но содержащие отдельные пробелы умение	В целом успешное, но содержащее отдельные пробелы применение навыков
5	Отл.	Сформированные систематические знания	Сформированное умение	Успешное и систематическое применение навыков

Шкала 2. Комплексная оценка сформированности знаний, умений и владений

Обозначения		Формулировка требований к степени сформированности компетенции
Цифр.	Оценка	
1	Неуд.	Не имеет необходимых представлений о проверяемом материале
2	Удовл. или неуд. (по усмотрению преподавателя)	Знать на уровне ориентирования, представлений. Субъект учения знает основные признаки или термины изучаемого элемента содержания, их отнесенность к определенной науке, отрасли или объектам, узнает их в текстах, изображениях или схемах и знает, к каким источникам нужно обращаться для более детального его усвоения.
3	Удовл.	Знать и уметь на репродуктивном уровне. Субъект учения знает изученный элемент содержания репродуктивно: произвольно воспроизводит свои знания устно, письменно или в демонстрируемых действиях.
4	Хор.	Знать, уметь, владеть на аналитическом уровне. Зная на репродуктивном уровне, указывать на особенности и взаимосвязи изученных объектов, на их достоинства, ограничения, историю и перспективы развития и особенности для разных объектов усвоения.
5	Отл.	Знать, уметь, владеть на системном уровне. Субъект учения знает изученный элемент содержания системно, произвольно и доказательно воспроизводит свои знания устно, письменно или в демонстрируемых действиях,

		учитывая и указывая связи и зависимости между этим элементом и другими элементами содержания учебной дисциплины, его значимость в содержании учебной дисциплины.
--	--	--

Индикаторные критерии оценки

Оценка	Индикаторные критерии оценки
Оценка «удовлетворительно»	<ul style="list-style-type: none"> • Реализована загрузка данных с клавиатуры. • Программа запускается и верно выполняет задание согласно варианту • Студент имеет представление о работе программы.
Оценка «хорошо»	<ul style="list-style-type: none"> • То же что и на оценку «удовлетворительно» с применением следующих элементов: • Реализованы функции записи и чтения информации в/из файл(а). • Использование динамической памяти (операция new). • Функции добавления или удаления записей в файле. • Функция изменения записей в файле. • Умение объяснить принципы работы разработанной программы.
Оценка «отлично»	<ul style="list-style-type: none"> • То же что и на оценку «хорошо», дополнительно • Использование классов и объектов для решения поставленной задачи согласно варианту с применением следующих элементов: • Конструкторов и деструкторов. • Дружбы классов. • Наследование (простое или множественное). • Перегрузка операций. • Виртуальные функции. • Реализовано шифрование и дешифрование файлов на основе системы CRYPTO++ или WinCrypt

Сводная таблица
критериев

	Оценка «удовлетворительн о»			Оценка "хорошо" тоже что и на оценку «удовлетворительно» с применением следующих элементов:					Оценка "отлично" тоже что и на оценку «хорошо», дополнительно: использование классов и объектов для решения поставленной задачи согласно варианту с применением следующих элементов:							
	Реализована загрузка данных с клавиатуры.	Программа запускается и верно выполняет задание согласно варианту	Студент имеет представление о работе	Реализованы функции записи и чтения информации в/из файл(а).	Использование динамической памяти	Функции добавления или удаления записей в файле.	Функция изменения записей в файле.	Умение объяснить принципы работы разработанной программы.	Конструкторов и деструкторов.	Друзья классов.	Наследование (простое или множественное).	Перегрузка операций.	Виртуальные функции.	Шифрование файла	Дешифрование файла	Умение объяснить работу любого элемента разработанной программы
балл за выпол нение	10	10	20	10	5	10	10	30	10	10	10	10	5	15	15	60
Оценк а	Балл					Набранное кол-во баллов			Пояснения							
оценк а 3	не менее					40			максимальная сумма баллов за "оценка удовлетворительно"							
оценк а 4	не менее					88,75			сумма баллов за "оценка удовлетворительно" и 75% от суммы баллов за оценку "хорошо"							
оценк а 5	не менее					213,75			сумма баллов за "оценка удовлетворительно" , баллов за оценку "хорошо" и 75% от суммы баллов за оценку "отлично"							

Варианты заданий

Вариант задания определяется по последним двум цифрам в зачетной книжке.

Вариант 1. Отсортировать группу по убыванию успеваемости любой одной или нескольких сессий (в т.ч. м.б. и всех), вводимых по желанию пользователя.

Вариант 2. Распечатать всех студентов, у которых за все время обучения нет ни одной оценки, а) 3 б) 3 и 4 в) 5 г) 3 и 5 д) 4 и 5 Варианты а)-д) выбираются по желанию пользователя. Их можно выбрать как 1, так и несколько или все варианты.

Вариант 3. Распечатать всех студентов, у которых за все время обучения не более 25% оценок а) 3 б) 3 и 4 в) 5 г) 3 и 5 д) 4 и 5. Варианты а-д выбираются по желанию пользователя. Их можно выбрать как 1, так и несколько или все варианты.

Вариант 4. Распечатать всех студентов в порядке убывания 5 в одном, нескольких или всех семестрах, которые выбираются по желанию пользователя.

Вариант 5. Отсортировать всех студентов в порядке уменьшения процентного содержания троек за один, несколько или все семестры, которые выбираются по желанию пользователя.

Вариант 6. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом в одной, нескольких или всех сессиях, выбираемых по желанию пользователя.

Вариант 7. Распечатать всех студентов, у которых за все время обучения нет ни одной тройки.

Вариант 8. Распечатать всех студентов, у которых не более 25 процентов троек за все время обучения.

Вариант 9. Распечатать в порядке убывания всех студентов, по количеству пятерок во 2-ом семестре.

Вариант 10. Отсортировать всех студентов в порядке уменьшения процентного содержания "троек" за 1 и 2 семестры.

Вариант 11. Отсортировать группу по уменьшению успеваемости любой сессии.

Вариант 12. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом.

Вариант 13. Отсортировать группу по убыванию успеваемости 2-ой сессии, или же вводимой по желанию пользователя

Вариант 14. Разбить группу на 2 части: мужскую и женскую. Отсортировать каждую часть по алфавиту.

Вариант 15. Разбить группу на 2 части:
хорошисты и отличники;
троечники.

Каждую часть отсортировать по номерам зачетных книжек.

Вариант 16. Разбить группу на 2 части:

1) по алфавиту от А до П;

2) по алфавиту от Р до Я.

Каждую часть отсортировать в порядке увеличения среднего бала за все время обучения.

Вариант 17. Разбить группу на 2 части:

1) 50 процентов хороших и отличных оценок за все время обучения;

2) Все остальные студенты.

Распечатать в каждой части 2-х наиболее успевающих и наиболее неуспевающих студентов.

Вариант 18. Разбить группу на 3 части:

1) отличников;

2) хорошистов;

3) троечников

по сессии, вводимой по желанию пользователя за все время обучения.

Вариант 19. Разбить группу на 3 части:

- 1) отличников;
- 2) хорошистов;
- 3) троечников

за все время обучения. Отсортировать каждую часть по алфавиту.

Вариант 20. Разбить группу на 3 части:

- 1) отличников;
- 2) отличников и хорошистов;
- 3) хорошистов и троечников.

за семестр вводимый по желанию пользователя. Распечатать по алфавиту" студентов каждой части группы.

Вариант 21. Разбить группу на 2 части:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Отсортировать каждую часть по номеру зачетных книжек.

Вариант 22. Разбить группу на 2 части:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Отсортировать каждую часть по алфавиту.

Вариант 23. Разбить группу на 2 части:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Отсортировать каждую часть по успеваемости за все время обучения.

Вариант 24. Разбить группу на 2 части:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Найти в каждой части наиболее успевающих и наиболее неуспевающих студентов.

Вариант 25. Отсортировать всю группу по увеличению года поступления в ВУЗ.

Вариант 26. Разбить группу на две части:

- 1) сдавших все спецпредметы только на 4 и 5
- 2) сдавших спецпредметы на 3,4,5.

Отсортировать каждую часть по алфавиту.

Вариант 27. Отсортировать группу по успеваемости в каждой сессии или вводимой по требованию .
пользователя

Вариант 28 Отсортировать группу по убыванию успеваемости любой одной или нескольких сессий (в т.ч. м.б. и всех), вводимых по желанию пользователя. С выбором пола человека

Вариант 29 Распечатать всех студентов, у которых за все время обучения нет ни одной оценки, а) 3 б) 3 и 4 в) 5 Предусмотреть распечатывать лиц определенного пола. Варианты а)-в) выбираются по желанию пользователя.

Вариант 30. Распечатать всех студентов, у которых за все время обучения не более 25% оценок а) 3 б) 3 и 4 в) 5 г) 3 и 5 д) 4 и 5. Варианты а-д выбираются по желанию пользователя. Предусмотреть распечатывать лиц определенного пола.

Вариант 31. Распечатать всех студентов в порядке убывания 5 в одном, нескольких или всех семестрах, которые выбираются по желанию пользователя. Предусмотреть распечатывать лиц определенного пола.

Вариант 32. Отсортировать всех студентов в порядке уменьшения процентного содержания троек за один, несколько или все семестры, которые выбираются по желанию пользователя. Предусмотреть осуществлять поиск лиц определенного пола.

Вариант 33. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом в одной, нескольких или всех сессиях, выбираемых по желанию пользователя. Предусмотреть осуществлять поиск среди лиц определенного пола

Вариант 34. Распечатать всех студентов, у которых за все время обучения нет ни одной тройки с поиском среди лиц определенного пола

Вариант 35. Распечатать всех студентов, у которых не более 25 процентов троек за все время обучения с поиском среди лиц определенного пола

Вариант 36. Распечатать в порядке убывания всех студентов, по количеству пятерок во 2-ом семестре с поиском лиц определенного пола

Вариант 37. Отсортировать всех студентов в порядке уменьшения процентного содержания "троек" за 1 и 2 семестры, с поиском среди лиц определенного пола

Вариант 38. Отсортировать группу по уменьшению успеваемости любой сессии, с поиском среди лиц определенного пола.

Вариант 39. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом, с поиском среди лиц определенного пола.

Вариант 40. Отсортировать группу по убыванию успеваемости 2-ой сессии, или же вводимой по желанию пользователя, с поиском среди лиц определенного пола

Вариант 41. Разбить группу на 2 части: мужскую и женскую. Отсортировать каждую часть по алфавиту.

Вариант 42. Разбить группу на 2 части, с поиском среди лиц определенного пола:
хорошисты и отличники;
троечники.

Каждую часть отсортировать по номерам зачетных книжек.

Вариант 43. Разбить группу на 2 части, с поиском среди лиц определенного пола:

- 1) по алфавиту от А до П;
- 2) по алфавиту от Р до Я.

Каждую часть отсортировать в порядке увеличения среднего бала за все время обучения.

Вариант 44. Разбить группу на 2 части, с поиском среди лиц определенного пола:

- 1) 50 процентов хороших и отличных оценок за все время обучения;
- 2) Все остальные студенты.

Распечатать в каждой части 2-х наиболее успевающих и наиболее неуспевающих студентов.

Вариант 45. Разбить группу на 3 части, с поиском среди лиц определенного пола:

- 1) отличников;
- 2) хорошистов;
- 3) троечников

по сессии, вводимой по желанию пользователя.

Вариант 46. Разбить группу на 3 части, с поиском среди лиц определенного пола:

- 1) отличников;
- 2) хорошистов;
- 3) троечников

за все время обучения. Отсортировать каждую часть по алфавиту.

Вариант 47. Разбить группу на 3 части, с поиском среди лиц определенного пола:

- 1) отличников;
- 2) отличников и хорошистов;
- 3) хорошистов и троечников.

за семестр вводимый по желанию пользователя. Распечатать по алфавиту" студентов каждой части группы.

Вариант 48. Разбить группу на 2 части, с поиском среди лиц определенного пола:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Отсортировать каждую часть по номеру зачетных книжек.

Вариант 49. Разбить группу на 2 части, с поиском среди лиц определенного пола:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Отсортировать каждую часть по алфавиту.

Вариант 50. Разбить группу на 2 части, с поиском среди лиц определенного пола:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Отсортировать каждую часть по успеваемости за все время обучения.

Вариант 51. Разбить группу на 2 части, с поиском среди лиц определенного пола:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Найти в каждой части наиболее успевающих и наиболее неуспевающих студентов.

Вариант 52. Отсортировать всю группу по увеличению года поступления в ВУЗ, с поиском среди лиц определенного пола.

Вариант 53. Разбить группу на две части, с поиском среди лиц определенного пола:

- 1) сдавших все спецпредметы только на 4 и 5
- 2) сдавших спецпредметы на 3,4,5.

Отсортировать каждую часть по алфавиту.

Вариант 54. Отсортировать группу по успеваемости в каждой сессии или вводимой по требованию .

пользователя, с поиском среди лиц определенного пола

Вариант 55. Отсортировать группу по убыванию успеваемости любой одной или нескольких сессий (в т.ч. м.б. и всех), вводимых по желанию пользователя, с указанием интервала года рождения.

Вариант 56. Распечатать всех студентов, у которых за все время обучения нет ни одной оценки а) 3 б) 3 и 4 в) 5 г) 3 и 5 д) 4 и 5. Их можно выбрать как 1, так и несколько или все варианты. Варианты а)-д) выбираются по желанию пользователя, с указанием интервала года рождения.

Вариант 57. Распечатать всех студентов, у которых за все время обучения не более 25% оценок а) 3 б) 3 и 4 в) 5 г) 3 и 5 д) 4 и 5. Варианты а-д выбираются по желанию пользователя. Их можно выбрать как 1, так и несколько или все варианты, с указанием интервала года рождения...

Вариант 58. Распечатать всех студентов в порядке убывания 5 в одном, нескольких или всех семестрах, которые выбираются по желанию пользователя, с указанием интервала года рождения.

Вариант 59. Отсортировать всех студентов в порядке уменьшения процентного содержания троек за один, несколько или все семестры, которые выбираются по желанию пользователя, с указанием интервала года рождения.

Вариант 60. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом в одной, нескольких или всех сессиях, выбираемых по желанию пользователя, с указанием интервала года рождения.

Вариант 61. Распечатать всех студентов, у которых за все время обучения нет ни одной тройки, с указанием интервала года рождения.

Вариант 62. Распечатать всех студентов, у которых не более 25 процентов троек за все время обучения, с указанием интервала года рождения.

Вариант 63. Распечатать в порядке убывания всех студентов, по количеству пятерок во 2-ом семестре, с указанием интервала года рождения.

Вариант 64. Отсортировать всех студентов в порядке уменьшения процентного содержания "троек" за 1 и 2 семестры, с указанием интервала года рождения.

Вариант 65. Отсортировать группу по уменьшению успеваемости любой сессии, с указанием интервала года рождения.

Вариант 66. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом, с указанием интервала года рождения.

Вариант 67. Отсортировать группу по убыванию успеваемости 2-ой сессии, или же вводимой по желанию пользователя, с указанием интервала года рождения.

Вариант 68. Разбить группу на 2 части: мужскую и женскую. Отсортировать каждую часть по алфавиту, с указанием интервала года рождения.

Вариант 69. Разбить группу на 2 части, с указанием интервала года рождения.:
хорошисты и отличники;
троечники.

Каждую часть отсортировать по номерам зачетных книжек.

Вариант 70. Разбить группу на 2 части, с указанием интервала года рождения.:

- 1) по алфавиту от А до П;
- 2) по алфавиту от Р до Я.

Каждую часть отсортировать в порядке увеличения среднего бала за все время обучения.

Вариант 71. Разбить группу на 2 части, с указанием интервала года рождения.:

- 1) 50 процентов хороших и отличных оценок за все время обучения;
- 2) Все остальные студенты.

Распечатать в каждой части 2-х наиболее успевающих и наиболее неуспевающих студентов, с указанием интервала года рождения.

Вариант 72. Разбить группу на 3 части, с указанием интервала года рождения.:

- 1) отличников;
- 2) хорошистов;
- 3) троечников

по сессии, вводимой по желанию пользователя за все время обучения.

Вариант 73. Разбить группу на 3 части, с указанием интервала года рождения.:

- 1) отличников;
- 2) хорошистов;
- 3) троечников

за все время обучения. Отсортировать каждую часть по алфавиту.

Вариант 74. Разбить группу на 3 части, с указанием интервала года рождения.:

- 1) отличников;
- 2) отличников и хорошистов;
- 3) хорошистов и троечников.

за семестр вводимый по желанию пользователя. Распечатать по алфавиту" студентов каждой части группы.

Вариант 75. Разбить группу на 2 части, с указанием интервала года рождения.:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
 - 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.
- Отсортировать каждую часть по номеру зачетных книжек.

Вариант 76. Разбить группу на 2 части, с указанием интервала года рождения.:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
 - 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.
- Отсортировать каждую часть по алфавиту.

Вариант 77. Разбить группу на 2 части, с указанием интервала года рождения.:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
 - 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.
- Отсортировать каждую часть по успеваемости за все время обучения.

Вариант 78. Разбить группу на 2 части, с указанием интервала года рождения.:

- 1) студентов, поступивших в ВУЗ в одном и том же году;
- 2) студентов, поступивших в ВУЗ в другие годы, отличные от части 1.

Найти в каждой части наиболее успевающих и наиболее неуспевающих студентов.

Вариант 79. Отсортировать всю группу по увеличению года поступления в ВУЗ, с указанием интервала года рождения.

Вариант 80. Разбить группу на две части, с указанием интервала года рождения.:

1)сдавших все спец предметы только на 4 и 5

2) сдавших спец предметы на 3,4,5.

Отсортировать каждую часть по алфавиту.

Вариант 81. Отсортировать группу по успеваемости в каждой сессии или вводимой по требованию.

пользователя, с указанием интервала года рождения.

Вариант 82 Отсортировать группу по убыванию успеваемости любой одной или нескольких сессий (в т.ч. м.б. и всех), вводимых по желанию пользователя. С указанием интервала года рождения.

Вариант 83 Распечатать всех студентов, у которых за все время обучения нет ни одной оценки а) 3 б) 3 и 4 в) 5 Предусмотреть распечатывать лиц с указанием интервала года рождения.. Варианты а)-в) выбираются по желанию пользователя.

Вариант 84. Распечатать всех студентов, у которых за все время обучения не более 25% оценок а) 3 б) 3 и 4 в) 5 г) 3 и 5 д) 4 и 5. Варианты а-д выбираются по желанию пользователя. Предусмотреть распечатывать лиц с указанием интервала года рождения..

Вариант 85. Распечатать всех студентов в порядке убывания 5 в одном, нескольких или всех семестрах, которые выбираются по желанию пользователя. Предусмотреть распечатывать лиц с указанием интервала года рождения..

Вариант 86. Отсортировать всех студентов в порядке уменьшения процентного содержания троек за один, несколько или все семестры, которые выбираются по желанию пользователя. Предусмотреть осуществлять поиск лиц с указанием интервала года рождения.

Вариант 87. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом в одной, нескольких или всех сессиях, выбираемых по желанию пользователя. Предусмотреть осуществлять поиск среди лиц с указанием интервала года рождения.

Вариант 88. Распечатать всех студентов, у которых за все время обучения нет ни одной тройки с поиском среди лиц с указанием интервала года рождения.

Вариант 89. Распечатать всех студентов, у которых не более 25 процентов троек за все время обучения с поиском среди лиц с указанием интервала года рождения.

Вариант 90. Распечатать в порядке убывания всех студентов, по количеству пятерок во 2-ом семестре с поиском лиц с указанием интервала года рождения.

Вариант 91. Отсортировать всех студентов в порядке уменьшения процентного содержания "троек" за 1 и 2 семестры, с поиском среди лиц с указанием интервала года рождения.

Вариант 92. Отсортировать группу по уменьшению успеваемости любой сессии, с поиском среди лиц с указанием интервала года рождения.

Вариант 93. Найти и распечатать все данные о студентах, которые успевают с наибольшим и наименьшим успехом, с поиском среди лиц с указанием интервала года рождения..

Вариант 94. Отсортировать группу по убыванию успеваемости 2-ой сессии, или же вводимой по желанию пользователя, с поиском среди лиц с указанием интервала года рождения.

Вариант 95. Разбить группу на 2 части: мужскую и женскую. Отсортировать каждую часть по алфавиту.

Вариант 96. Разбить группу на 2 части, с поиском среди лиц с указанием интервала года рождения.:

хорошисты и отличники;

троечники.

Каждую часть отсортировать по номерам зачетных книжек.

Вариант 97. Разбить группу на 2 части, с поиском среди лиц с указанием интервала года рождения.:

1) по алфавиту от А до П;

2) по алфавиту от Р до Я.

Каждую часть отсортировать в порядке увеличения среднего бала за все время обучения.

Вариант 98. Разбить группу на 2 части, с поиском среди лиц с указанием интервала года рождения.:

- 1) 50 процентов хороших и отличных оценок за все время обучения;
- 2) Все остальные студенты.

Распечатать в каждой части 2-х наиболее успевающих и наиболее неуспевающих студентов.

Вариант 99. Разбить группу на 3 части, с поиском среди лиц с указанием интервала года рождения.:

- 1) отличников;
- 2) хорошистов;
- 3) троечников

по сессии, вводимой по желанию пользователя.

Вариант 00. Разбить группу на 3 части, с поиском среди лиц с указанием интервала года рождения.:

- 1) отличников;
- 2) хорошистов;
- 3) троечников

за все время обучения. Отсортировать каждую часть по алфавиту.

Необходимые сведения из теории

Основные сведения из языков программирования

Массив символов

Символьная переменная (переменная типа `char`) - это величина размером в 1 байт, которая используется для представления литер и целых чисел в диапазоне от 0 до 255 или от -128 до 127, в зависимости от того, знаковая эта переменная или беззнаковая. Символьные константы заключаются в одинарные кавычки. Примеры символьных констант: `'a'`, `'+'`, `'1'`.

Символьная константа `"` - это символ с нулевым значением, так называемый символ `NULL`. Вместо просто 0 часто используют запись `"`, чтобы подчеркнуть символьную природу используемого выражения.

Строковая константа, или строковый литерал - это нуль или более символов, заключенных в двойные кавычки, например,

```
"это строковая константа"  
"" /*это пустая строка (нуль)*/.
```

Таким образом, в языке C строка (строковая константа) - это массив символов, заканчивающийся нулевым байтом. Поэтому памяти для строки требуется на один байт больше, чем число символов, расположенных между двойными кавычками.

Следующие инициализации массивов эквивалентны:

```
char* str="hello";  
char str[6]={ 'h','e','l','l','o',' '};
```

Символьные строки и функции работы со строками

Всякий раз, когда компилятор встречается с чем-то заключённым в кавычки, он определяет это как строковую константу. Символы `'\0'` записываются в последовательные ячейки памяти. Если необходимо включить кавычки в символьную строку, нужно поставить впереди `'\"'`.

Строковые константы размещаются в области данных. Вся фраза в кавычках является указателем на место в памяти, где записана строка. Это аналогично имени массива, служащего указателем на адрес расположения массива. Для вывода символьной константы служит идентификатор `%s`.

```
char mas[] = "Это одна строка";  
mas - адрес строки -> &m[0] *mas=="Э".
```

Можно использовать указатель для создания строки.

```
char *str = "Таблица результатов";  
char str[] = "Таблица результатов".
```

Сама строка размещается в области данных, а указатель инициализируется адресом.

```
void main(void){  
    char* mesg="Ошибка чтения";  
    char* copy;  
    copy = mesg; создается второй указатель на строку.  
    printf ("%S", copy);  
    printf ("%S", mesg);  
}  
char* name;  
scanf ("%S", name); //Так нельзя !!! Указателю не присвоен адрес.  
char name [81]; //Нужно в начале определить массив
```

Массивы символьных строк

1. Строки в символьный массив можно вводить с клавиатуры.

```
char mas[80];  
scanf("%S",mas);
```

2. Если требуется несколько строк, то организуем цикл ввода

```
char mas[4][81];
for (i=0; i<4; i++)
    scanf("%S", mas[i]);           // &mas[i][0]
```

3. Можно сразу инициализировать в программе.

```
char m1[] = "Только одна строка"; // автоматически определяется
// длина строки + 1 байт на '\0'.
```

4. Размер массива можно задать явно.

```
char m2[50] = "Только одна строка"; //18+1
```

5. `char m3[]={ 'c', 'm', 'p', 'o', 'k', 'a', '\0' };`

6. Инициализация массива строк:

```
char masstr[3][16]={ "Первая строка",
    "Вторая строка",
    "Третья строка" };
*masstr[0]=='П';
*masstr[1]=='В';
*masstr[2]=='Т';
```

7. «Рванный массив» – это массив указателей на строки.

```
static char *masstr[3]= { "Первая строка",
    "Вторая строка",
    "Третья строка" };
```

В случае «рваного массива» длина строк разная и зря не расходуется память.

Массивы указателей

Можно определять массивы указателей

```
int* parray[5];           //5 указателей на целые значения.
*parray[3] -             //3-й элемент массива.
char *keywords[5]={ "ADD", "CHANGE", "DELETE", "LIST", "QUIT" };
В памяти
```

```
keyword[0] – адрес 10000 строка ADD\0 46
keyword[1] - 10004 CHANGE\0 76
keyword[2] - 10011 DELETE\0 76
for (i=0; i<5; i++)
    printf("%S", keywords[i]);
char *key[3],**pt;        //определение указателя на указатель
pt=key;
printf(«%s %d\n»,*pt,**pt); //распечатывается первая строка и код
//первой буквы
```

Функции, работающие со строками

Функции, определенные в заголовочном файле `stdio.h`.

1. Функция `gets(char *)` - вводит строку в массив с клавиатуры

```
char name[81];
gets(name); // Берёт все символы до конца строки символ '\n'
// отбрасывает и записывает '\0' и передаёт в вызов программы.
```

2. Функция `puts(char *)` - выводит строку на экран

```
puts ("Я функция puts()");
char str1[]="Только одна строка";
puts(str1); //Читает строку до встречи '\0'. Все строки
//выводятся с новой строки.
```

3. Функция `int getc(stdin)` – вводит символ в переменную с клавиатуры.

4. Функция `int putc(int c, stdout)` – выводит символ на экран.

Стандартные библиотечные функции

Функции, определенные в заголовочном файле `string.h`.

1. `int strlen(char *)` - определяет длину строки без `'\0'`;
`int k = strlen(str1);`

2. `char * strcat(char *, char *)` - объединяет две строки в одну.
`strcat(str1, str2);` //результат в первом массиве.

```
void main (void){
    char str1[80]="Мой любимый цветок";
    char str2[10]="ромашка";
    if (strlen(str1)+strlen(str2)< 80-1)
        strcat(str1, str2);    //Чтобы копировать - необходимо проверить,
//чтобы длины массива было достаточно на
//две строки + ноль-байт.
}
```

3. `char * strncat(char *, char *,int)` - объединяет 1-ю строку и n-байтов второй строки в одну.

4. `int strcmp(char *, char *)` - сравнение строк.

```
#define ANSWER "YES"
void main(void){
    char try[10];
    gets(try);
    puts ("Вы студенты 203 группы?");
    while (strcmp(try, ANSWER)!=0){
        puts ("Попробуйте ещё раз")
        gets (try);
    }
    puts("Верно");
}
```

Функция возвращает 0, если строки одинаковы. Сравнение идёт до признака конца строки - `'\0'`, а не до конца массива, или до первого несравнения:

В-А возвращает 1, А-В возвращает -1.

5. `int strncmp(char *, char *,int)` - сравнение n байт у 2-х строк .

6. `char * strcpy(char *, char *)` - копирование строк

```
#define WORD "Таблица результатов"
void main (void){
    char str1[30]; //Длина массива не проверяется
    strcpy(str1, WORD)
    puts(str1);
}
```

7. `char *strcpy(char *, char *,int)` - копирование n байт строки
8. `char *strdup(char *)` – выделяет память и копирует строку.
9. `char *strupr(char *)` – преобразует строчные буквы в прописные.
10. `char *strlwr(char *)` – преобразует прописные буквы в строчные.
11. `char *strrev(char *)` – реверсирует строку.
12. `char *strchr(char *, int)` – устанавливает позицию первого вхождения символа
14. `char *strrchr(char *, int c)` – устанавливает позицию последнего вхождения символа с.
15. `char *strstr(char *, char *)` - устанавливает позицию первого вхождения подстроки str2 в строку str1.
16. `int stricmp(char *, char *)` – сравнивает не различая строчные и прописные буквы.

Преобразование символьных строк

Функции, определённые в заголовочном файле `stdlib.h`.

1. `int atoi()` - строку в целое.
`double atof()` - строку в число с плавающей точкой.

```
void main(void){
char num[10];
int val;
puts("Введите число");
gets(num);
val=atoi(num); // обрабатывает до 1-го символа не являющегося
}                // цифрой
```

2. Существуют функции обратного преобразования числа в строку.

`itoa(int val, char *str, int radix)` - целое в строку, где
`int val` – число;
`char *str` – строка;
`int radix` – система счисления.

`ltoa(long val, char *str, int radix)` - с плавающей точкой в строку.

Функции, определённые в заголовочном файле `ctype.h`.

Выполняют преобразования только с буквами английского алфавита.

1. Преобразование строчной буквы в прописную - `int toupper(int c)`
2. Проверка буква прописная или нет - `int isupper(int c)`
3. Преобразование прописной буквы в строчную - `int tolower(int c)`
4. Проверка буква строчная или нет - `int islower(int c)`

```
#include <ctype.h>
void main(void){
int ch; crit=0;        //Признак прописные или строчные буквы
while ((ch=getche())!='\n'){
    if(crit==0){
        ch=isupper(ch) ? tolower(ch): ch;
        putchar(ch);
    }
    else{
        ch=islower(ch)? toupper(ch):ch;
        putchar (ch);
    }
}
```

Ссылки

Ссылка - это псевдоним для другой переменной. Они объявляются при помощи символа &. Ссылки должны быть проинициализированы при объявлении, причем только один раз.

Тип "ссылка на type" определяется следующим образом:

type& имя_перем.

Ссылка при определении сразу же инициализируется. Инициализация ссылки производится следующим образом:

```
int i = 0;
```

```
int& iref = i;
```

Физически iref представляет собой постоянный указатель на int - переменную типа int* const. Ее значение не может быть изменено после ее инициализации. Ссылка отличается от указателя тем, что используется не как указатель, а как переменная, адресом которой она была инициализирована:

```
iref++; //то же самое, что i++
```

```
int *ip = &iref; //то же самое, что ip = &i;
```

Таким образом, iref становится синонимом переменной i.

При передаче больших объектов в функции с помощью ссылки он не копируется в стек и, следовательно, повышается производительность.

```
#include <iostream.h>
```

```
void incr (int&);
```

```
void main(void){
```

```
    int i = 5;
```

```
    incr(i);
```

```
    cout<< "i= " << i << "\n";
```

```
}
```

```
void incr (int& k){
```

```
    k++;
```

```
}
```

Поскольку ссылка — это псевдоним, то при передаче объекта в функцию по ссылке внутри нее объект можно изменять. Ссылки не могут ссылаться на другие ссылки или на поле битов.

Не может быть массивов ссылок или указателей на ссылку.

Ссылка может использоваться для возврата результата из функции. Возвратить результат по ссылке - значит вернуть не указатель на объект и не его значение, а сам этот объект.

Работа с памятью. Указатели

Рассмотрим такую задачу: в файле записаны целые числа. Надо отсортировать их и записать в другой файл. Проблема заключается в том, что заранее неизвестно, сколько в файле таких чисел. В такой ситуации есть два варианта:

Выделить память с запасом, если известно, например, что этих чисел гарантированно не более 1000.

Использовать динамическое выделение памяти – сначала определить, сколько чисел в массиве, а потом выделить ровно столько памяти, сколько нужно.

Наиболее грамотным решением является второй вариант (использование динамических массивов). Для этого надо сначала поближе познакомиться с указателями.

Указатель – это переменная, в которой хранится адрес другой переменной или участка памяти.

Указатели являются одним из основных понятий языка Си. В такие переменные можно записывать адреса любых участков памяти, на чаще всего – адрес начального элемента динамического массива. Что же можно делать с указателями?

Объявить	<code>char *pC; int *pI; float *pF;</code>	Указатели объявляются в списке переменных, но перед их именем ставится знак *. Указатель всегда указывает на переменную того типа, для которого он был объявлен. Например, pC может указывать на любой символ, pI – на любое целое число, а pF – на любое вещественное число.
Присвоить адрес	<code>int i, *pI; ... pI = &i;</code>	Такая запись означает: «записать в указатель pI адрес переменной i». Запись &i обозначает адрес переменной i.
Получить значение по этому адресу	<code>float f, *pF; ... f = *pF;</code>	Такая запись означает: «записать в переменную f то вещественное число, на которое указывает указатель pF». Запись *pF обозначает содержимое ячейки, на которую указывает pF.
Сдвинуть	<code>int *pI; ... pI++; pI += 4; -- pI; pI -= 4;</code>	В результате этих операций значение указателя меняется особым способом: pI++ сдвигает указатель на РАЗМЕР ЦЕЛОГО ЧИСЛА, то есть на 4 байта, а не на 1 как можно подумать. А запись pI+=4 или pI=pI+4 сдвигает указатель на 4 целых числа дальше, то есть на 16 байт.
Обнулить	<code>char *pC; ... pC = NULL;</code>	Если указатель равен нулевому адресу NULL, это обычно означает, что указатель недействительный. По нему нельзя ничего записывать (это вызывает сбой программы компьютера).
Вывести на экран	<code>int i, *pI; ... pI = &i; printf("Адр.i=%p", pI);</code>	Для вывода указателей используется формат %p.

Итак, что надо знать про указатели:

- указатель – это переменная, в которой записан адрес другой переменной;
- при объявлении указателя надо указать тип переменных, на которых он будет указывать, а перед именем поставить знак *;
- знак & перед именем переменной обозначает ее адрес;
- знак * перед указателем в рабочей части программы (не в объявлении) обозначает значение ячейки, на которую указывает указатель;
- нельзя записывать по указателю, который указывает непонятно куда – это вызывает сбой программы, поскольку что-то стирается в памяти;
- для обозначения недействительного указателя используется константа NULL;
- при изменении значения указателя на n он в самом деле сдвигается к n-ому следующему числу данного типа, то есть для указателей на целые числа на $n * \text{sizeof}(\text{int})$ байт;
- указатель печатаются по формату %p.

Динамическое выделение памяти

Динамическими называются массивы, размер которых неизвестен на этапе написания программы. Прием, о котором будем говорить, относится уже не к стандартному языку Си, а к его расширению Си ++. Существуют и стандартные способы выделения памяти в языке Си (с помощью функций malloc и calloc), но они не очень удобны.

Следующая простейшая программа, которая использует динамический массив, вводит с клавиатуры размер массива, все его элементы, а затем сортирует их и выводит на экран.

```
#include <stdio.h> main()
{

    int N;          // размер массива (заранее неизвестен)
    int *A; // указатель для выделения памяти
    printf ("Размер массива > "); // ввод размера массива
    scanf ("%d", &N);
    A = new int [N]; // выделение памяти
    if ( A == NULL ) { // если не удалось выделить
        printf("Не удалось выделить память");
        return 1; // выход по ошибке, код ошибки 1
    }
    for (i = 0; i < N; i ++ ) { // дальше так же, как для массива
        printf ("\nA[%d] > ", i+1);
        scanf ("%d", &A[i]);
    }
    // здесь сортировка и вывод на экран
    delete A; // освободить память
}
```

Итак, мы хотим выделить в памяти место под новый массив целых чисел во время работы программы. Мы уже знаем его размер, пусть он хранится в переменной N (это число обязательно должно быть больше нуля). Оператор выделения памяти new вернет нам адрес нового выделенного блока, и для работы с массивом нам надо где-то его запомнить. Вы уже знаете, что есть специальный класс переменных для записи в них адресов памяти – указатели.

- динамические массивы используются тогда, когда на момент написания программы размер массива неизвестен
- для того, чтобы работать с динамическим массивом, надо объявить указатель соответствующего типа (в нем будет храниться адрес первого элемента массива);
- `int *A;`
- когда требуемый размер массива стал известен, надо использовать оператор new языка Си++, указав в скобках размер массива (в программе для краткости нет проверки на положительность N) ;
- `A = new int[N];`
- нельзя использовать оператор new при отрицательном или нулевом N;
- после выделения памяти надо проверить, успешно ли оно прошло; если память выделить не удалось, то значение указателя будет равно NULL, использование такого массива приведет к сбою программы;
- работа с динамическим массивом, на который указывает указатель A, идет также, как и с обычным массивом размера N (помните, что язык Си не следит за выходом за границы массива);
- после использования массива надо освободить выделенную память, вызвав оператор
- `delete A;`
- после освобождения памяти значение указателя не изменяется, но использовать его уже нельзя, потому что память освобождена;
- учитывая, что при добавлении числа к указателю он сдвигается на заданное число ячеек ЗАДАННОГО ТИПА, то следующие записи равносильны и вычисляют адрес i-ого элемента массива:
- `&A[i]` и `A+i`

Отсюда следует, что A совпадает с `&A[0]`, и поэтому имя массива можно использовать как адрес его начального элемента;

Ошибки, связанные с выделением памяти

Самые тяжелые и трудно выявляемые ошибки в программах на языке Си связаны именно с неверным использованием динамических массивов. В таблице перечислены наиболее тяжелые случаи и способы борьбы с ними.

Ошибка	Причина и способ лечения
Запись в чужую область памяти	Память была выделена неудачно, а массив используется. Вывод: надо всегда делать проверку указателя на NULL.
Повторное удаление	Массив уже удален и теперь удаляется снова.

указателя	Вывод: если массив удален из памяти, обнулите указатель – ошибка быстрее выявится.
Выход за границы массива	Запись в массив в элемент с отрицательным индексом или индексом, выходящим за границу массива
Утечка памяти	Неиспользуемая память не освобождается. Если это происходит в функции, которая вызывается много раз, то ресурсы памяти скоро будут израсходованы. Вывод: убирайте за собой «мусор» (освобождайте память).

Выделение памяти для матрицы

Для выделения памяти под одномерный массив целых чисел нам потребовался указатель на целые числа. Для матрицы надо выделить указатель на массив целых чисел, который объявляется как

```
int **A;
```

Но лучше всего сразу объявить новый тип данных - указатель на целое число. Новые типы объявляются директивой typedef вне всех процедур и функций (там же, где и глобальные переменные).

```
typedef int *pInt;
```

Этой строкой мы сказали компилятору, что любая переменная нового типа pInt представляет собой указатель на целое число или адрес массива целых чисел. К сожалению, место для матрицы не удастся так же просто выделить в памяти, как мы делали это для одномерного массива.

Если написать просто

```
int M = 5, N = 7;
```

```
pInt *A;
```

```
A = new int[M][N]; // ошибочная строка
```

компилятор выдает множество ошибок. Связано это с тем, что ему требуется заранее знать длину одной строки, чтобы правильно расшифровать запись типа A[i][j]. Ниже рассмотрены три способа решения этой проблемы.

Известный размер строки

Если размер строки матрицы известен, а неизвестно только количество строк, можно поступить так: ввести новый тип данных – строка матрицы. Когда количество строк станет известно, с помощью оператора new выделяем массив таких данных.

```
typedef int row10[10]; // новый тип: массив из 10 элементов
```

```
main()
```

```
{
```

```
int N;
```

```
row10 *A; // указатель на массив (матрица)
```

```
printf("Введите число строк "); scanf("%d", &N);
```

```
A = new row10[N]; // выделить память на N строк
```

```
A[0][1] = 25; // используем матрицу, как обычно
```

```
printf("%d", A[2][3]);
```

```
delete A; // освобождаем память
```

```
}
```

Неизвестный размер строки

Пусть размеры матрицы M и N заранее неизвестны и определяются в ходе работы программы. Тогда можно предложить следующий способ выделения памяти под новую матрицу. Поскольку матрицу можно рассматривать как массив из строк-массивов, объявим M указателей и выделим на каждый из них область памяти для одномерного массива размером N (то есть, на одну строку). Сами эти указатели тоже надо представить в виде динамического массива. Определив требуемые размеры матрицы, мы выделяем сначала динамический массив указателей, а потом на каждый указатель – место для одной строки.

```
typedef int *pInt;    // новый тип данных: указатель на целое

main()
{
    int M, N, i;

    pInt *A;          // указатель на указатель

    // ввод M и N
    A = new pInt[M];   // выделить память под массив указателей

    for ( i = 0; i < M; i ++ ) // цикл по всем указателям
        A[i] = new int[N];    // выделяем память на строку i

    // работаем с матрицей A, как обычно

    for ( i = 0; i < M; i ++ ) // освобождаем память для всех строк
        delete A[i];
    delete A;          // освобождаем массив указателей
}
```

В рассмотренном выше случае на каждую строку выделяется свой участок памяти. Можно поступить иначе: сначала выделим область памяти сразу на всю матрицу и запишем ее адрес в $A[0]$. Затем расставим указатели так, чтобы $A[1]$ указывал на $N+1$ -ый элемент с начала блока (начало строки 1), $A[2]$ – на $2N+1$ -ый (начало строки 2) и т.д. Таким образом, в памяти выделяется всего два блока – массив указателей и сама матрица.

```
typedef int *pInt; main()
{
    int M, N, i;

    pInt *A;          // указатель на указатель

    // ввод M и N
    A = new pInt[M];   // память на массив указателей

    A[0] = new int [M*N]; // память для матрицы
    for ( i = 1; i < M; i ++ ) // расставляем указатели
        A[i] = A[i-1] + N;

    // работаем с матрицей
    delete A[0];      // освобождаем матрицу
    delete A;          // освобождаем указатели
}
```

Стандартные функции работы с файлами.

Файл – это именованная область ячеек памяти, в которой хранятся данные одного типа. Файл имеет следующие характерные особенности:

- уникальное имя;
- однотипность данных;
- произвольная длина, которая ограничивается только емкостью диска.

Файлы бывают текстовыми и двоичными.

Текстовый файл – файл, в котором каждый символ из используемого набора хранится в виде одного байта (код, соответствующий символу). Текстовые файлы разбиваются на несколько строк с помощью специального символа "конец строки". Текстовый файл заканчивается специальным символом "конец файла".

Двоичный файл – файл, данные которого представлены в бинарном виде. При записи в двоичный файл символы и числа записываются в виде последовательности байт (в своем внутреннем двоичном представлении в памяти компьютера).

Все операции ввода-вывода реализуются с помощью функций, которые находятся в библиотеке C++. Библиотека C++ поддерживает три уровня ввода-вывода:

- потоковый ввод-вывод;
- ввод-вывод нижнего уровня;
- ввод-вывод для консоли и портов (зависит от ОС).

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Функции библиотеки ввода-вывода языка C++, поддерживающие обмен данными с файлами на уровне потока, позволяют обрабатывать данные различных размеров и форматов, обеспечивая при этом буферизованный ввод и вывод. Таким образом, поток представляет собой этот файл вместе с предоставленными средствами буферизации.

Чтение данных из потока называется извлечением, вывод в поток – помещением (включением).

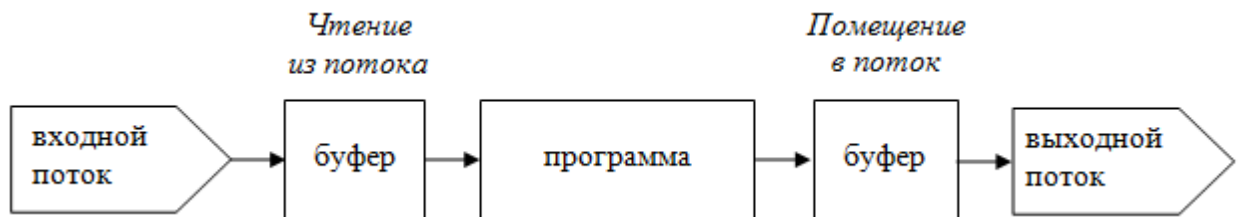


Рис. 1 Буферизация данных при работе с потоками

Для работы с файлом в языке C++ необходима ссылка на файл. Для определения такой ссылки существует структура FILE, описанная в заголовочном файле stdio.h. Данная структура содержит все необходимые поля для управления файлами, например: текущий указатель буфера, текущий счетчик байтов, базовый адрес буфера ввода-вывода, номер файла.

Функция открытия файла

При открытии файла (потока) в программу возвращается указатель на поток (файловый указатель), являющийся указателем на объект структурного типа FILE. Этот указатель идентифицирует поток во всех последующих операциях.

Например:

```
#include  
.....  
FILE *fp;
```

Для открытия файла существует функция fopen, которая инициализирует файл.

Синтаксис:

```
fp=fopen(ИмяФайла, РежимОткрытия);  
где fp – указатель на поток (файловый указатель);
```

ИмяФайла – указатель на строку символов, представляющую собой допустимое имя файла, в которое может входить спецификация файла (включает обозначение логического устройства, путь к файлу и собственно имя файла);

РежимОткрытия – указатель на строку режима открытия файла.

Например:

```
fp=fopen("t.txt","r");
```

Существуют несколько режимов открытия файлов.

Поток можно открыть в текстовом (t) или двоичном (b) режиме. По умолчанию используется текстовый режим. В явном виде режим указывается следующим образом:

"r+b" или "rb" – двоичный (бинарный) режим;

"r+t" или "rt" – текстовый режим.

Функция закрытия файла

Открытые на диске файлы после окончания работы с ними рекомендуется закрыть явно. Это является хорошим тоном в программировании.

Синтаксис:

```
int fclose(УказательНаПоток);
```

Возвращает 0 при успешном закрытии файла и -1 в противном случае.

Открытый файл можно открыть повторно (например, для изменения режима работы с ним) только после того, как файл будет закрыт с помощью функции fclose().

Функция удаления файла

Синтаксис:

```
int remove(const char *filename);
```

Эта функция удаляет с диска файл, указатель на который хранится в файловой переменной filename. Функция возвращает ненулевое значение, если файл невозможно удалить.

Функция переименования файла

Синтаксис:

```
int rename(const char *oldfilename, const char *newfilename);
```

Функция переименовывает файл; первый параметр – старое имя файла, второй – новое. Возвращает 0 при неудачном выполнении.

Функция контроля конца файла

Для контроля достижения конца файла есть функция feof.

```
int feof(FILE * filename);
```

Функция возвращает ненулевое значение, если достигнут конец файла.

Функции ввода-вывода данных файла

1) Символьный ввод-вывод

Для символьного ввода-вывода используются функции:

```
int fgetc(FILE *fp);
```

где fp – указатель на поток, из которого выполняется считывание.

Функция возвращает очередной символ в формате int из потока fp. Если символ не может быть прочитан, то возвращается значение EOF.

```
int fputc(int c, FILE*fp);
```

где fp – указатель на поток, в который выполняется запись;

c – переменная типа int, в которой содержится записываемый в поток символ.

Функция возвращает записанный в поток `fp` символ в формате `int`. Если символ не может быть записан, то возвращается значение `EOF`.

Пример 1.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
int main()
{
    FILE *f;
    int c;
    char *filename="t.txt";
    if ((f=fopen(filename,"r"))==0)
        perror(filename);
    else
        while((c = fgetc(f)) !=EOF)
            putchar(c);
    //вывод с на стандартное устройство вывода
    fclose(f);
    system("pause");
    return 0;
}
```

2) Строковый ввод-вывод

Для построчного ввода-вывода используются следующие функции:

```
char *fgets(char *s, int n, FILE *f);
```

где `char *s` – адрес, по которому размещаются считанные байты;

`int n` – количество считанных байтов;

`FILE *f` – указатель на файл, из которого производится считывание.

Прием байтов заканчивается после передачи `n-1` байтов или при получении управляющего символа `'\\n'`. Управляющий символ тоже передается в принимающую строку. Строка в любом случае заканчивается `'\\0'`. При успешном завершении считывания функция возвращает указатель на прочитанную строку, при неуспешном – `0`.

```
int fputs(char *s, FILE *f);
```

где `char *s` – адрес, из которого берутся записываемые в файл байты;

`FILE *f` – указатель на файл, в который производится запись.

Символ конца строки (`'\\0'`) в файл не записывается. Функция возвращает `EOF`, если при записи в файл произошла ошибка, при успешной записи возвращает неотрицательное число.

Пример 2. Построчное копирование данных из файла `f1.txt` в файл `f2.txt`.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
#define MAXLINE 255 //максимальная длина строки
```



```

int main()
{
    //копирование файла in в файл out
    FILE *in, //исходный файл
    *out; //принимающий файл
    char buf[MAXLINE];
    //строка, с помощью которой выполняется копирование
    in=fopen("f1.txt","r");
    //открыть исходный файл для чтения
    out=fopen("f2.txt","w");
    //открыть принимающий файл для записи
    while(fgets(buf, MAXLINE, in)!=0)
    //прочитать байты из файла in в строку buf
    fputs(buf, out);
    //записать байты из строки buf в файл out
    fclose(in); //закрыть исходный файл
    fclose(out); //закрыть принимающий файл
    system("pause");
    return 0;
}

```

Динамические структуры данных

Часто в серьезных программах надо использовать данные, размер и структура которых должны меняться в процессе работы. Динамические массивы здесь не выручают, поскольку заранее нельзя сказать, сколько памяти надо выделить – это выясняется только в процессе работы. Например, надо проанализировать текст и определить, какие слова и в каком количестве в нем встречаются, причем эти слова нужно расставить по алфавиту.

В таких случаях применяют данные особой структуры, которые представляют собой отдельные элементы, связанные с помощью ссылок.

Каждый элемент (узел) состоит из двух областей памяти: поля данных и ссылок. Ссылки – это адреса других узлов этого же типа, с которыми данный элемент логически связан. В языке Си для организации ссылок используются переменные указатели. При добавлении нового узла в такую структуру выделяется новый блок памяти и (с помощью ссылок) устанавливаются связи этого элемента с уже существующими. Для обозначения конечного элемента в цепи используются нулевые ссылки (NULL).

Линейный список

В простейшем случае каждый узел содержит всего одну ссылку. Для определенности будем считать, что решается задача частотного анализа текста – определения всех слов, встречающихся в тексте и их количества. В этом случае область данных элемента включает строку (длиной не более 40 символов) и целое число.



Каждый элемент содержит также ссылку на следующий за ним элемент. У последнего в списке элемента поле ссылки содержит NULL. Чтобы не потерять список, мы должны где-то (в переменной) хранить адрес его первого узла – он называется «головой» списка. В программе надо объявить два новых типа данных – узел списка Node и указатель на него PNode. Узел представляет собой структуру, которая содержит три поля - строку, целое число и указатель на такой же узел. Правилами языка Си допускается объявление

```

struct Node {
    char word[40]; // область данных

```

```

int count;
Node *next; // ссылка на следующий узел
}; typedef Node *PNode; // тип данных: указатель на узел

```

В дальнейшем мы будем считать, что указатель Head указывает на начало списка, то есть, объявлен в виде

```
PNode Head = NULL;
```

Первая буква «P» в названии типа PNode происходит от слова pointer – указатель (англ.) В начале работы в списке нет ни одного элемента, поэтому в указатель Head записывается нулевой адрес NULL.

Создание элемента списка

Для того, чтобы добавить узел к списку, необходимо создать его, то есть выделить память под узел и запомнить адрес выделенного блока. Будем считать, что надо добавить к списку узел, соответствующий новому слову, которое записано в переменной NewWord. Составим функцию, которая создает новый узел в памяти и возвращает его адрес. Обратите внимание, что при записи данных в узел используется обращение к полям структуры через указатель.

```

PNode CreateNode ( char NewWord[] )
{
    PNode NewNode = new Node; // указатель на новый узел  strcpy(NewNode->word,
NewWord); // записать слово  NewNode->count = 1; // счетчик слов = 1  NewNode-
>next = NULL; // следующего узла нет  return NewNode; // результат функции – адрес
узла }

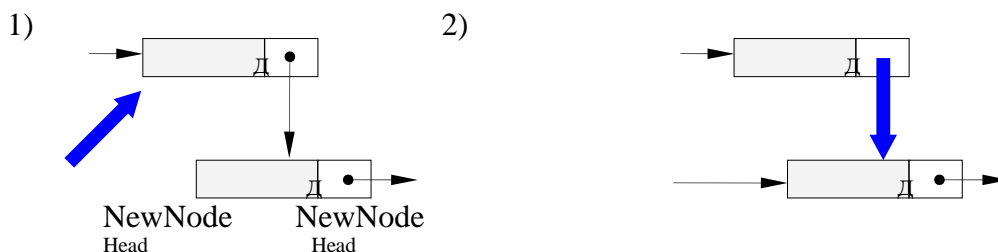
```

После этого узел надо добавить к списку (в начало, в конец или в середину).

Добавление узла

Добавление узла в начало списка

При добавлении нового узла NewNode в начало списка надо 1) установить ссылку узла NewNode на голову существующего списка и 2) установить голову списка на новый узел.



По такой схеме работает процедура AddFirst. Предполагается, что адрес начала списка хранится в Head. Важно, что здесь и далее адрес начала списка передается по ссылке, так как при добавлении нового узла он изменяется внутри процедуры.

```

void AddFirst (PNode &Head, PNode NewNode)
{
    NewNode->next = Head;  Head = NewNode;
}

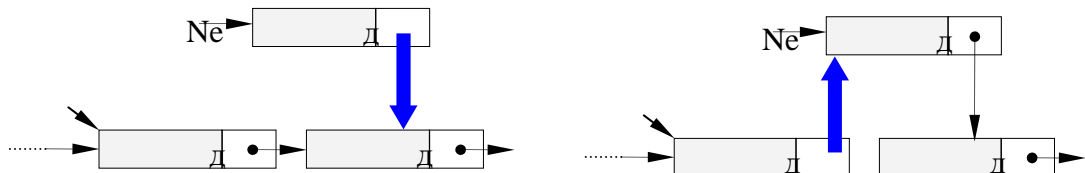
```

Добавление узла после заданного

Дан адрес NewNode нового узла и адрес p одного из существующих узлов в списке. Требуется вставить в список новый узел после узла с адресом p. Эта операция выполняется в два этапа:

1) установить ссылку нового узла на узел, следующий за данным; 2) установить ссылку данного узла p на NewNode.

1) 2)



Последовательность операций менять нельзя, потому что если сначала поменять ссылку у узла p, будет потерян адрес следующего узла.

```
void AddAfter (PNode p, PNode NewNode)
{
    NewNode->next = p->next;
    p->next = NewNode;
}
```

Добавление узла перед заданным

Эта схема добавления самая сложная. Проблема заключается в том, что в простейшем линейном списке (он называется односвязным, потому что связи направлены только в одну сторону) для того, чтобы получить адрес предыдущего узла, нужно пройти весь список сначала. Задача сведется либо к вставке узла в начало списка (если заданный узел – первый), либо к вставке после заданного узла.

```
void AddBefore(PNode &Head, PNode p, PNode NewNode)
{
    PNode q = Head;
    if (Head == p) {
        AddFirst(Head, NewNode); // вставка перед первым узлом    return;
    }
    while (q && q->next!=p) // ищем узел, за которым следует p    q = q->next;
    if ( q )                // если нашли такой узел,
        AddAfter(q, NewNode); //    добавить новый после него }
```

Такая процедура обеспечивает «защиту от дурака»: если задан узел, не присутствующий в списке, то в конце цикла указатель q равен NULL и ничего не происходит.

Существует еще один интересный прием: если надо вставить новый узел NewNode до заданного узла p, вставляют узел после этого узла, а потом выполняется обмен данными между узлами NewNode и p. Таким образом, по адресу p в самом деле будет расположен узел с новыми данными, а по адресу NewNode – с теми данными, которые были в узле p, то есть мы решили задачу. Этот прием не сработает, если адрес нового узла NewNode запоминается где-то в программе и потом используется, поскольку по этому адресу будут находиться другие данные.

Добавление узла в конец списка

Для решения задачи надо сначала найти последний узел, у которого ссылка равна NULL, а затем воспользоваться процедурой вставки после заданного узла. Отдельно надо обработать случай, когда список пуст.

```
void AddLast(PNode &Head, PNode NewNode)
{
    PNode q = Head;
    if (Head == NULL) {        // если список пуст,
        AddFirst(Head, NewNode); // вставляем первый элемент    return;
    }
    while (q->next) q = q->next; // ищем последний элемент
    AddAfter(q, NewNode); }
```

Проход по списку

Для того, чтобы пройти весь список и сделать что-либо с каждым его элементом, надо начать с головы и, используя указатель next, продвигаться к следующему узлу.

```
PNode p = Head;    // начали с головы списка while ( p != NULL ) { // пока не дошли до конца
```

```
    // делаем что-нибудь с узлом p
```

```
    p = p->next;    // переходим к следующему узлу }
```

Поиск узла в списке

Часто требуется найти в списке нужный элемент (его адрес или данные). Надо учесть, что требуемого элемента может и не быть, тогда просмотр заканчивается при достижении конца списка. Такой подход приводит к следующему алгоритму:

начать с головы списка;

пока текущий элемент существует (указатель – не NULL), проверить нужное условие и перейти к следующему элементу;

закончить, когда найден требуемый элемент или все элементы списка просмотрены.

Например, следующая функция ищет в списке элемент, соответствующий заданному слову (для которого поле word совпадает с заданной строкой NewWord), и возвращает его адрес или NULL, если такого узла нет.

```
PNode Find (PNode Head, char NewWord[])
```

```
{
```

```
    PNode q = Head;
```

```
    while (q && strcmp(q->word, NewWord))    q = q->next;
```

```
    return q; }
```

Вернемся к задаче построения алфавитно-частотного словаря. Для того, чтобы добавить новое слово в нужное место (в алфавитном порядке), требуется найти адрес узла, перед которым надо вставить новое слово. Это будет первый от начала списка узел, для которого «его» слово окажется «больше», чем новое слово. Поэтому достаточно просто изменить условие в цикле while в функции Find., учитывая, что функция strcmp возвращает «разность» первого и второго слова.

```
PNode FindPlace (PNode Head, char NewWord[])
```

```
{
```

```
    PNode q = Head;
```

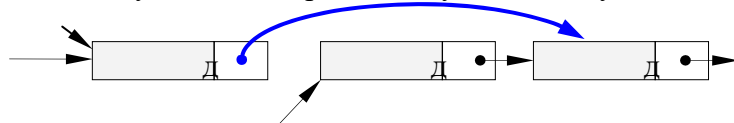
```
    while (q && (strcmp(q->word, NewWord) > 0))
```

```
        q = q->next;    return q; }
```

Эта функция вернет адрес узла, перед которым надо вставить новое слово (когда функция strcmp вернет положительное значение), или NULL, если слово надо добавить в конец списка.

Удаление узла

Эта процедура также связана с поиском заданного узла по всему списку, так как нам надо поменять ссылку у предыдущего узла, а перейти к нему непосредственно невозможно. Если мы нашли узел, за которым идет удаляемый узел, надо просто переставить ссылку.



OldNode

Отдельно обрабатывается случай, когда удаляется первый элемент списка. При удалении узла освобождается память, которую он занимал.

Отдельно рассматриваем случай, когда удаляется первый элемент списка. В этом случае адрес удаляемого узла совпадает с адресом головы списка Head и надо просто записать в Head адрес следующего элемента.

```
void DeleteNode(PNode &Head, PNode OldNode)
{
    PNode q = Head; if (Head == OldNode)
        Head = OldNode->next; // удаляем первый элемент else
    {
        while (q && q->next != OldNode) // ищем элемент
            q = q->next;
        if ( q == NULL ) return; // если не нашли, выход    q->next = OldNode->next;
    }
    delete OldNode;          // освобождаем память }

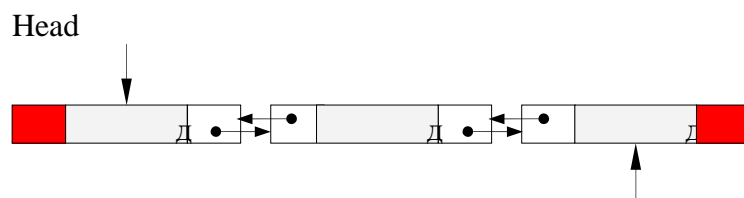
```

Барьеры

Для рассмотренного варианта списка требуется отдельно обрабатывать граничные случаи: добавление в начало, добавление в конец, удаление одного из крайних элементов. Можно значительно упростить приведенные выше процедуры, если установить два барьера – фиктивные первый и последний элементы. Таким образом, в списке всегда есть хотя бы два элемента-барьера, а все рабочие узлы находятся между ними.

Двусвязный список

Многие проблемы при работе с односвязным списком вызваны тем, что в них невозможно перейти к предыдущему элементу. Возникает естественная идея – хранить в памяти ссылку не только на следующий, но и на предыдущий элемент списка. Для доступа к списку используется не одна переменная-указатель, а две – ссылка на «голову» списка (Head) и на «хвост» - последний элемент (Tail).



Каждый узел содержит (кроме полезных данных) также ссылку на следующий за ним узел (поле next) и предыдущий (поле prev). Поле next у последнего элемента и поле prev у первого содержат NULL. Узел объявляется так:

```
struct Node {
    char word[40]; // область данных    int count;
    Node *next, *prev; // ссылки на соседние узлы
};
typedef Node *PNode; // тип данных «указатель на узел»

```

В дальнейшем мы будем считать, что указатель Head указывает на начало списка, а указатель Tail – на конец списка:

```
PNode Head = NULL, Tail = NULL;
```

Для пустого списка оба указателя равны NULL.

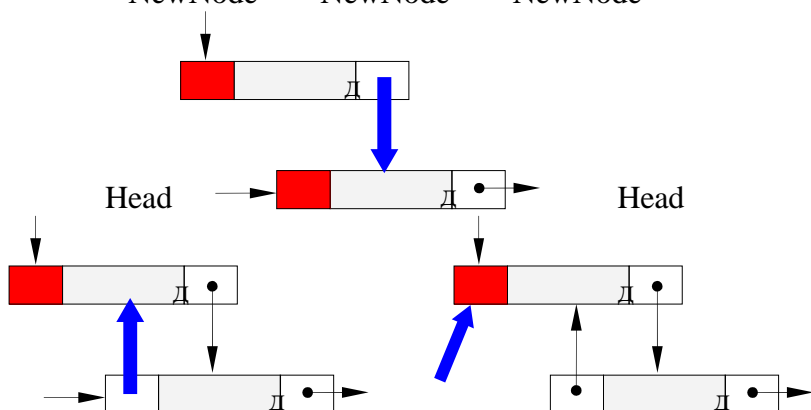
Операции с двусвязным списком

Добавление узла в начало списка

При добавлении нового узла NewNode в начало списка надо установить ссылку next узла NewNode на голову существующего списка и его ссылку prev в NULL;

установить ссылку prev бывшего первого узла (если он существовал) на NewNode; 3)
установить голову списка на новый узел;
4) если в списке не было ни одного элемента, хвост списка также устанавливается на
новый узел.

1) NewNode 2) NewNode 3) NewNode



По такой схеме работает следующая процедура:

```
void AddFirst(PNode &Head, PNode &Tail, PNode NewNode)
{
    NewNode->next = Head;  NewNode->prev = NULL;
    if ( Head ) Head->prev = NewNode;  Head = NewNode;
    if ( ! Tail ) Tail = Head; // этот элемент – первый }
```

Добавление узла в конец списка

Благодаря симметрии добавление нового узла NewNode в конец списка проходит совершенно аналогично, в процедуре надо везде заменить Head на Tail и наоборот, а также поменять prev и next.

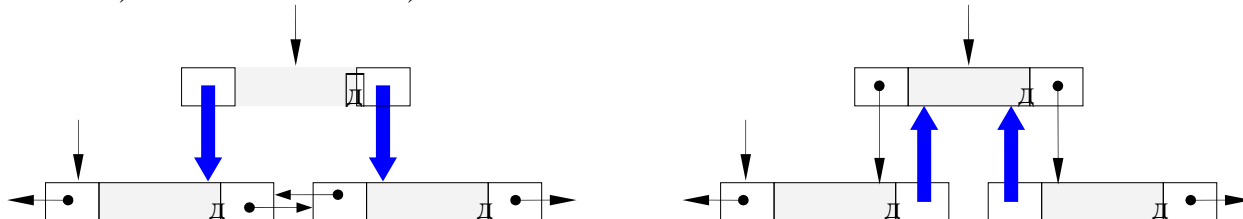
Добавление узла после заданного

Дан адрес NewNode нового узла и адрес p одного из существующих узлов в списке. Требуется вставить в список новый узел после p. Если узел p является последним, то операция сводится к добавлению в конец списка (см. выше). Если узел p – не последний, то операция вставки выполняется в два этапа:

установить ссылки нового узла на следующий за данным (next) и предшествующий ему (prev);

установить ссылки соседних узлов так, чтобы включить NewNode в список.

1) NewNode 2) NewNode



Такой метод реализует приведенная ниже процедура (она учитывает также возможность вставки элемента в конец списка, именно для этого в параметрах передаются ссылки на голову и хвост списка):

```
void AddAfter (PNode &Head, PNode &Tail,
              PNode p, PNode NewNode)
{
    if ( ! p->next )
        AddLast (Head, Tail, NewNode); // вставка в конец списка else {
```

```

NewNode->next = p->next; // меняем ссылки нового узла
NewNode->prev = p;
p->next->prev = NewNode; // меняем ссылки соседних узлов
p->next = NewNode;
}
}

```

Добавление узла перед заданным выполняется аналогично.

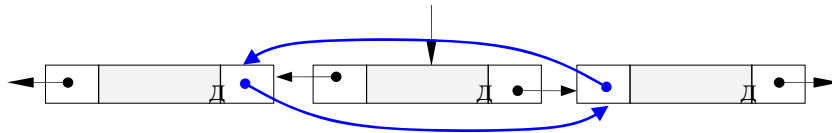
Поиск узла в списке

Проход по двусвязному списку может выполняться в двух направлениях – от головы к хвосту (как для односвязного) или от хвоста к голове.

Удаление узла

Эта процедура также требует ссылки на голову и хвост списка, поскольку они могут измениться при удалении крайнего элемента списка. На первом этапе устанавливаются ссылки соседних узлов (если они есть) так, как если бы удаляемого узла не было бы. Затем узел удаляется и память, которую он занимает, освобождается. Эти этапы показаны на рисунке внизу. Отдельно проверяется, не является ли удаляемый узел первым или последним узлом списка.

OldNode



```

void Delete(PNode &Head, PNode &Tail, PNode OldNode)
{
if (Head == OldNode) {
    Head = OldNode->next; // удаляем первый элемент if ( Head )
    Head->prev = NULL;
else Tail = NULL;      // удалили единственный элемент
} else {
    OldNode->prev->next = OldNode->next; if ( OldNode->next )
    OldNode->next->prev = OldNode->prev;
else Tail = NULL;      // удалили последний элемент
}
delete OldNode;
}

```

Циклические списки

Иногда список (односвязный или двусвязный) замыкают в кольцо, то есть указатель next последнего элемента указывает на первый элемент, и (для двусвязных списков) указатель prev первого элемента указывает на последний. В таких списках понятие «хвоста» списка не имеет смысла, для работы с ним надо использовать указатель на «голову», причем «головой» можно считать любой элемент.

Парадигмы ООП – инкапсуляция, наследование, полиморфизм

Инкапсуляция - сведение кода и данных воедино в одном объекте, получившем название класс.

Наследование - наличие в языке ООП механизма, позволяющего объектам класса наследовать характеристики более простых и общих типов. Наследование обеспечивает как требуемый уровень общности, так и необходимую специализацию.

Полиморфизм - дословный перевод с греческого "много форм". В С++ полиморфизм реализуется с помощью виртуальных функций, которые позволяют в рамках всей иерархии классов иметь несколько версий одной и той же функции. Решение о том, какая именно версия должна выполняться в данный момент, определяется на этапе выполнения программы и носит название позднего связывания.

Существует несколько реализаций системы, поддерживающих стандарт С++, из которых можно выделить реализации Visual С++ (Microsoft) и Builder С++ (Inprise). Отличия относятся в основном к используемым библиотекам классов и средам разработки. В действительности в С++ программах можно использовать библиотеки языка С, библиотеки классов С++, библиотеки визуальных классов VCL (Builder С++), библиотеку MFC (Visual С++ и Builder С++).

Язык С++ является родоначальником множества объектно-ориентированных языков, таких как Java, С#, РНР и др.

Использование динамических массивов. Теоретические сведения

Объявление динамического массива

Массивы, создаваемые в динамической памяти, будем называть *динамическими* (размерность становится известна в процессе выполнения программы). При описании массива после имени в квадратных скобках задается количество его элементов (размерность), например `int a[10]`. Размерность массива может быть задана только константой или константным выражением.

При описании массив можно инициализировать, то есть присвоить его элементам начальные значения, например:

```
int a[10] = {1, 1, 2, 2, 5, 100};
```

Если инициализирующих значений меньше, чем элементов в массиве, остаток массива обнуляется, если больше — лишние значения не используются. Элементы массивов нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности. Номер элемента указывается после его имени в квадратных скобках, например, `a[0]`, `a[3]`.

Если до начала работы программы неизвестно, сколько в массиве элементов, в программе следует использовать динамические массивы. Память под них выделяется с помощью операции `new` или функции `malloc` в динамической области памяти во время выполнения программы. Адрес начала массива хранится в переменной, называемой указателем. Например:

```
int n = 10;  
int *mass1 = new int[n];
```

Во второй строке описан указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью операции `new`. Выделяется столько памяти, сколько необходимо для хранения `n` величин типа `int`. Величина `n` может быть переменной. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементу обычного. Если динамический массив в какой-то момент работы программы перестает быть нужным и мы собираемся впоследствии использовать эту память повторно, необходимо освободить ее с помощью операции `delete[]`, например: `delete [] a;` (размерность массива при этом не указывается).

```
delete[] mass1;
```

При необходимости создания многомерных динамических массивов сначала необходимо с помощью операции `new` выделить память под `n` указателей (вектор, элемент которого - указатель), при этом все указатели располагаются в памяти последовательно друг за другом. После этого необходимо в цикле каждому указателю присвоить адрес выделенной области памяти размером, равным второй границе массива

```
mass2=new int*[row];  
// mass2 - указатель на массив указателей на одномерные массивы
```



```

for(i=0;i<row;i++)
mass2[i]=new int[col]; // каждый элемент массива указывает на одномерный
for (i=0; i<row;i++)
for (j=0;j<col;j++)

```

Освобождение памяти от двумерного динамического массива:

```

for(i=0;i<row;i++) //удаление всех одномерных
delete[] mass2[i]; // массивов
delete[] mass2;    // удаление массива указателей на одномерные массивы

```

Классы. Программирование линейных алгоритмов с использованием функций инициализации set() и вывода результатов print(). Теоретические сведения

Основное отличие C++ от C состоит в том, что в C++ имеются классы. С точки зрения языка C классы в C++ - это структуры, в которых вместе с данными определяются функции. Это и есть инкапсуляция в терминах ООП.

Класс (class) - это тип, определяемый пользователем, включающий в себя данные и функции, называемые методами или функциями-членами класса.

Данные класса - это то, что класс знает.

Функции-члены (методы) класса - это то, что класс делает.

Таким образом, определение типа задаваемого пользователем (class) содержит спецификацию данных, требующихся для представления объекта этого типа, и набор операций (функций) для работы с подобными объектами.

Объявление класса

Приведем пример объявления класса

```

class my_Fun
{
// компоненты-данные
double x,y;
// компоненты-функции
public:
// функция инициализации
void set(char *c,double X)
{
x=X;
y=sin(x);
}
// функция вывода результатов
void print(void)
{
cout << point<<y << endl;
}
};

```

Обычно описания классов включают в заголовочные файлы (*.H), а реализацию функций-членов классов - в файлы *.CPP.

Для каждого объекта класса устанавливается область видимости либо явно – указанием уровня доступа одним из ключевых слов public, private, protected с двоеточием, либо неявно – по умолчанию. Указание области видимости относится ко всем последующим объектам класса, пока не встретится указание другой области видимости. Область видимости public разрешает доступ к объектам класса из любой части программы, в которой известен этот объект (общедоступный). Область видимости private разрешает доступ к объектам класса только из методов этого класса. Объекты с такой областью видимости называют

частными. Область видимости `protected` определяется для защищенных объектов, она имеет смысл только в иерархической системе классов и разрешает доступ к объектам этой области из методов производных классов. В теле класса ключевое слово области видимости может использоваться неоднократно. Область видимости для объектов типа «класс» по умолчанию `private`.

Способы объявления и инициализации объектов и доступ к методам класса:

1. Прямой вызов

```
my_Fun Fun1; //объявление объекта1,но не инициализация
Fun1.set("Function1 = ",1.0); // инициализация данных
Fun1.print();           // прямой вызов
cout << "Input enter1..." << endl<<endl;
```

2. Косвенный вызов

```
my_Fun *p1 = &Fun1; // воспользовались объектом 1
                // новая инициализация
p1->set("Function1 = ",1.0); // косвенный вызов
p1->print();           // косвенный вызов
cout << "Input enter1..." << endl<<endl;
```

3. Динамическое выделение памяти

```
my_Fun *p1 = new my_Fun;
p1->set("Function1 = ",1.0); // косвенный вызов
p1->print();           // косвенный вызов
cout << "Input enter1..." << endl<<endl;
// удаляется динамически выделенный объект
delete p1;
```

Классы. Программирование линейных алгоритмов с использованием конструктора, деструктора, friend - функции инициализации `set()` и функции вывода результатов `print()`. Теоретические сведения

Конструктор класса

Конструктор – это метод класса, имя которого совпадает с именем класса.

Конструктор вызывается автоматически после выделения памяти для переменной и обеспечивает инициализацию компонент – данных. Конструктор не имеет никакого типа (даже типа `void`) и не возвращает никакого значения в результате своей работы. Конструктор нельзя вызывать как обычную компонентную функцию в программе. Для класса может быть объявлено несколько конструкторов, различающихся числом и типами параметров. При этом даже если для объектного типа не определено ни одного конструктора, компилятор создает для него конструктор по умолчанию, не использующий параметров, а также конструктор копирования, необходимый в том случае, если переменная объектного типа передается в конструктор как аргумент. В этом случае создаваемый объект будет точной копией аргумента конструктора.

```
class my_Fun
{
// компоненты-данные
double x;
unsigned size;
public:
// объявление конструктора 1 (с параметрами)
my_Fun (double X=0);
// объявление конструктора 2 (без параметров)
my_Fun(void);
```

```

// объявление и описание деструктора
~my_Fun ()
{
cout<<"Destroyed object... "<<endl;
}
// описание конструктора 1
my_Fun::my_Fun (double X)
{
cout<<"Constructor1...."<<endl;
x=X;
}
// описание конструктора 2
my_Fun::my_Fun (void)
{
cout<<"Constructor2..."<<endl;
x=5.0;
}
}

```

Деструктор класса

Еще одним специальным методом класса является деструктор. Деструктор вызывается перед освобождением памяти, занимаемой объектной переменной, и предназначен для выполнения дополнительных действий, связанных с уничтожением объектной переменной, например, для освобождения динамической памяти, закрытия, уничтожения файлов и т.п. Деструктор всегда имеет то же имя, что и имя класса, но перед именем записывается знак ~ (тильда). Деструктор не имеет параметров и подобно конструктору не возвращает никакого значения. Таким образом, деструктор не может быть перегружен и должен существовать в классе в единственном экземпляре. Деструктор вызывается автоматически при уничтожении объекта. Таким образом, для статически определенных объектов деструктор вызывается, когда заканчивается блок программы, в котором определен объект (блок в данном случае – составной оператор или тело функции). Для объектов, память для которых выделена динамически, деструктор вызывается при уничтожении объекта операцией delete.

Дружественная функция (friend)

В языке C++ одна и та же функция не может быть компонентом двух разных классов. Чтобы предоставить функции возможность выполнения действий над различными классами можно определить обычную функцию языка C++ и предоставить ей право доступа к элементам класса типа private, protected. Для этого нужно в описании класса поместить заголовок функции, перед которым поставить ключевое слово friend. Дружественная функция не является методом класса, не зависит от позиции в классе и спецификаторов прав доступа. Friend – функции получают доступ к членам класса через указатель, передаваемый им явно. Можно сделать все функции класса Y друзьями класса X в одном объявлении.

Класс «Динамическая строка» и перегрузка операций. Теоретические сведения

Для представления символьной (текстовой) информации можно использовать символы, символьные переменные и символьные константы.

Символьная константа представляется последовательностью символов, заключенной в кавычки: "Начало строки \n". В C++ нет отдельного типа для строк. Массив символов - это и есть строка. Количество элементов в таком массиве на один элемент больше, чем изображение строки, т. к. в конец строки добавлен '\0' (нулевой байт).

Присвоить значение массиву символов с помощью обычного оператора присваивания нельзя. Поместить строку в массив можно либо при вводе, либо с помощью инициализации:

```
char s[] = "ABCDEF";
```

Для работы со строками существует специальная библиотека string.h. Примеры функций для работы со строками из библиотеки string.h в таблице:

Функции работы со строками

Функция	Прототип и краткое описание функции
strcmp	int strcmp(const char *str1, const char *str2); Сравнивает строки str1 и str2. Если str1 < str2, то результат отрицательный, если str1 = str2, то результат равен 0, если str1 > str2, то результат положительный.
strcpy	char* strcpy(char*s1, const char *s2); Копирует байты из строки s1 в строку s2
strdup	char *strdup (const char *str); Выделяет память и переносит в нее копию строки str.
strlen	unsigned strlen (const char *str); Вычисляет длину строки str.
strncat	char *strncat(char *s1, const char *s2, int kol); Приписывает kol символов строки s1 к строке s2.
strncpy	char *strncpy(char *s1, const char *s2, int kol); Копирует kol символов строки s1 в строку s2.
strnset	char *strnset(char *str, int c, int kol); Заменяет первые kol символов строки s1 символом c.

Строки, при передаче в функцию, в качестве фактических параметров могут быть определены либо как одномерные массивы типа char[], либо как указатели типа char*. В отличие от обычных массивов в этом случае нет необходимости явно указывать длину строки.

Функции преобразования строки S в число:

целое: int atoi(S); длинное целое: long atol(S); действительное: double atof(S); при ошибке возвращает значение 0.

Функции преобразования числа V в строку S:

целое: itoa(int V, char S, int kod); длинное целое: ltoa(long V, char S, int kod);
2 <= kod <= 36, для отрицательных чисел kod = 10.

Перегрузка операций

Для перегрузки операции для класса в C++ используется следующий синтаксис:

```
<Тип> operator <операция>(<входные параметры>)
{
<операторы>;
}
```

где < Тип > - тип, возвращаемый функцией;

operator - ключевое слово;

< операция > - перегружаемая операция.

В языке C++ имеются следующие ограничения на перегрузку операций:

- C++ не различает префиксную и постфиксную формы ++ и --;
- переопределяемая операция должна присутствовать в языке (например, нельзя определить операцию с символом #);
- нельзя переопределить операторы, заданные следующими символами . * :: ? ;
- переопределённые операции сохраняют свой изначальный приоритет.

Наличие в классе конструктора String:: String(String&) и операторов присваивания позволяет защитить объекты класса от побитового копирования.

Пример: Ввести с клавиатуры строку символов. Признак окончания ввода строки - нажатие клавиши "Ввод". Программа должна определить длину введенной строки L и если L < 10 – вернуть строку, которая не содержит заглавных латинских букв.

```
#include <iostream.h>
```

```
#define SIZE 255 //длина строки по умолчанию
```

```

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <istream.h>
class X{
char *str;
char *str_return;
public:
X(); //конструктор по-умолчанию
X(char*); //конструктор, которому можно передавать параметр
~X(); //деструктор
char* Run(); //метод, выполняющий поставленную задачу.
void Set(char*);
friend void print(X&); //функция-друг печати
friend ostream& operator<<(ostream&,X&); //перегрузка оператора вывода
friend istream& operator>>(istream&,X&); //перегрузка оператора ввода
friend char* Run(X&); //функция-друг, выполняющий поставленную задачу.
};
X::X(){
str=new char[SIZE];
str[0]='\0';
str_return=new char[SIZE];
str_return[0]='\0';
};
X::X(char *s){
str=new char[SIZE];
strcpy(str,s);
str_return=new char[SIZE];
str_return[0]='\0';
};
X::~~X(){
delete[] str;
cout<<"...destructor has been called"<<endl;
};
void X::Set(char* s){
for (unsigned int i=0;i<strlen(s);i++)
str[i]=s[i];
str[i]='\0';
};
char* X::Run(){ /*метод, решающий конкретную задачу, в данном случае - выделение
из строки подстроки, не содержащей заглавных латинских букв, если длина исходной строки
меньше 10*/
int j=0;
if (strlen(str)<10) {
for (unsigned int i=0;i<strlen(str);i++)
if ( ((int)str[i]<65) || ((int)str[i]>90) ) {
str_return[j]=str[i]; j++;
};
str_return[j]='\0';
}
else strcpy(str_return,str);

```

```

return str_return;
};
char* Run(X &obj){return obj.Run();};
void print(X &obj){cout<<obj.str<<" "<<obj.str_return<<endl;};
ostream& operator<<(ostream &stream,X &ob) {
stream << ob.str ;
return stream;
};
istream &operator>>(istream &stream,X &ob){
stream >> ob.str;
return stream;
};
void main (void){
char s[265];

cout<<"Type anything and press \"Enter\":"<<endl;
cin.getline(s,256); //считываем полностью всю строку
X str(s); //доступ к методам класса непосредственно через переменную,
//начальное значение устанавливаем через конструктор
cout<<"You have type:"<<endl;
print(str);
cout<<"Output string:"<<endl;
cout<<Run(str)<<endl;
cout<<"Type anything and press \"Enter\":"<<endl;
cin.getline(s,256);
X *pstr; //доступ к методам класса через указатель
pstr=new X();
pstr->Set(s);
cout<<"You have type:"<<endl;
print(*pstr);
cout<<"Output string:"<<endl;
cout<<Run(*pstr)<<endl;
delete pstr;
};

```

Наследование классов, механизм виртуальных функций. Теоретические сведения

Наследование - механизм создания производного класса из базового. Т.е., к существующему классу можно что-либо добавить, или изменять его каким-либо образом для создания нового (производного) класса. Это мощный механизм для повторного использования кода. Наследование позволяет создавать иерархию связанных типов, совместно использующих код и интерфейс. Модификатор прав доступа используется для изменения доступа к наследуемым объектам в соответствии с правилами, указанными в таблице

Доступ в классах при наследовании

Доступ в базовом классе	Модификатор прав доступа	Доступ в производном классе
private	private	не доступны
private	public	не доступны

protected	private	private
protected	public	protected
public	private	private
public	public	public

Ограничение на наследование

При определении производного класса не наследуются из базового:

1. конструкторы;
2. деструкторы;
3. операторы new, определенные пользователем;
4. операторы присвоения, определенные пользователем;
5. отношения дружественности.

Использование косвенной адресации с установкой указателей на базовый класс.

Механизм косвенной адресации рассмотрим на примере:

```
class B
{
public:
int x;
B() {          // Конструктор по умолчанию
x = 4; }
};
class D : public B {    // Производный класс
public:
int y;
D()
{          // Конструктор по умолчанию
y = 5; }
};
void main(void) {
D d; // Конструктор класса D создает объект d
B *p; // Указатель установлен на базовый класс
p = &d; // Указатель p инициализируется адресом d
// косвенное обращение к объектам базового и производного классов
// «считываем их текущее состояние в переменные
int i = p -> x; // Базовый класс виден напрямую
int j = ( ( D* ) p )->y; // Прямое преобразование указателя на D
// через переменные печатаем их текущее состояние
cout << "x_i=" << i << endl;
cout << "y_j=" << j << endl;
getch();
}
```

Виртуальная функция и механизм позднего связывания

Виртуальная функция объявляется в базовом или производном классе и, затем, переопределяется в наследуемых классах. Совокупность классов (подклассов), в которых определяется и переопределяется виртуальная функция, называется полиморфическим кластером, ассоциированным с некоторой виртуальной функцией. В пределах полиморфического кластера сообщение связывается с конкретной виртуальной функцией-методом во время выполнения программы.

Обычную функцию-метод можно переопределить в наследуемых классах. Однако без атрибута `virtual` такая функция-метод будет связана с сообщением на этапе компиляции. Атрибут `virtual` гарантирует позднее связывание в пределах полиморфического кластера.

Часто возникает необходимость передачи сообщений объектам, принадлежащим разным классам в иерархии. В этом случае требуется реализация механизма позднего связывания. Чтобы добиться позднего связывания для объекта, его нужно объявить как указатель или ссылку на объект соответствующего класса. Для открытых производных классов указатели и ссылки на объекты этих классов совместимы с указателями и ссылками на объекты базового класса (т.е. к объекту производного класса можно обращаться, как будто это объект базового класса). Выбранная функция-метод зависит от класса, на объект которого указывается, но не от типа указателя.

C++ поддерживает `virtual` функции-методы, которые объявлены в основном классе и переопределены в порожденном классе. Иерархия классов, определенная общим наследованием, создает связанный набор типов пользователя, на которые можно ссылаться с помощью указателя базового класса. При обращении к виртуальной функции через этот указатель в C++ выбирается соответствующее функциональное определение во время выполнения. Объект, на который указывается, должен содержать в себе информацию о типе, поскольку различия между ними может быть сделано динамически. Это особенность типична для ООП кода. Каждый объект “знает” как на него должны воздействовать. Эта форма полиморфизма называется чистым полиморфизмом.

В C++ функции-методы класса с различным числом и типом параметров есть действительно различные функции, даже если они имеют одно и то же имя. Виртуальные функции позволяют переопределять в управляемом классе функции, введенные в базовом классе, даже если число и тип аргументов то же самое. Для виртуальных функций нельзя переопределять тип функции. Если две функции с одинаковым именем будут иметь различные аргументы, C++ будет считать их различными и проигнорирует механизм виртуальных функций. Виртуальная функция обязательно метод класса.

Программирование шаблона классов. Теоретические сведения

Достаточно часто встречаются классы, объекты которых должны содержать элементы данных произвольного типа (в том смысле, что их тип определяется отдельно для каждого конкретного объекта). В качестве примера можно привести любую структуру данных (массив указателей, массив, список, дерево). Для этого в C++ предлагаются средства, позволяющие определить некоторое множество идентичных классов с параметризованным типом внутренних элементов. Они представляют собой особого вида заготовку класса, в которой в виде параметра задан тип (класс) входящих в него внутренних элементов данных. При создании конкретного объекта необходимо дополнительно указать и конкретный тип внутренних элементов в качестве фактического параметра. Создание объекта сопровождается созданием соответствующего конкретного класса для типа, заданного в виде параметра. Принятый в C++ способ определения множества классов с параметризованным внутренним типом данных (иначе, макроопределение) называется шаблоном (template).

Синтаксис шаблона рассмотрим на примере шаблона класса векторов, содержащих динамический массив указателей на переменные заданного типа.

```
// <class T> - параметр шаблона - класс "T", внутренний тип данных
// vector - имя группы шаблонных классов
template <class T> class vector
{
    int  tsize;    // Общее количество элементов
    int  csize;    // Текущее количество элементов
    T    **obj;    // Массив указателей на парам. объекты типа "T"
public:
    T *operator[](int); // оператор [int] возвращает указатель на
    // параметризованный объект класса "T"
```



```
void insert(T*); // включение указателя на объект типа "T"
int index(T*);
};
```

Данный шаблон может использоваться для порождения объектов-векторов, каждый из которых хранит объекты определенного типа. Имя класса при этом составляется из имени шаблона "vector" и имени типа данных (класса), который подставляется вместо параметра "T":

```
vector<int> a;
vector<double> b;
extern class time;
vector<time> c;
```

Заметим, что транслятором при определении каждого вектора с новым типом объектов генерируется описание нового класса по заданному шаблону (естественно, неявно в процессе трансляции). Например, для типа `int` транслятор получит:

```
class vector<int>
{
int tsize;
int csize;
int **obj;
public:
int *operator[](int);
void insert(int*);
int index(int*);
};
```

Далее следует очевидное утверждение, что функции-методы шаблона также должны быть параметризованы, то есть генерироваться для каждого нового типа данных. Действительно, это так: функции-методы шаблона классов в свою очередь также являются шаблонными функциями с тем же самым параметром. То же самое касается переопределяемых операторов:

```
// параметр шаблона - класс "T", внутренний тип данных
// имя функции-элемента или оператора - параметризовано
//
template <class T> T* vector<T>::operator[](int n)
{
if (n >= tsize) return(NULL);
return (obj[n]);
}
template <class T> int vector<T>::index(T *pobj)
{
int n;
for (n=0; n<tsize; n++)
if (pobj == obj[n]) return(n);
return(-1);
}
```

Заметим, что транслятором при определении каждого вектора с новым типом объектов генерируется набор методов-функций по заданным шаблонам (естественно, неявно

в процессе трансляции). При этом сами шаблонные функции должны размещаться в том же заголовочном файле, где размещается определение шаблона самого класса. Для типа `int` сгенерированные транслятором функции-методы будут выглядеть так:

```
int* vector<int>::operator[](int n)
{
    if (n >= tsize) return(NULL);
    return (obj[n]);
}
int vector<int>::index(int *pobj)
{
    int n;
    for (n=0; n<tsize; n++)
        if (pobj == obj[n]) return(n);
    return(-1);
}
```

Множественное наследование с использованием абстрактных базовых классов, файлового ввода-вывода с применением потоков C++, функций обработки исключительных ситуаций. Теоретические сведения

Абстрактные классы

Если базовый класс используется только для порождения производных классов, то виртуальные функции в базовом классе могут быть "пустыми", поскольку никогда не будут вызваны для объекта базового класса. Базовый класс в котором есть хотя бы одна такая функция, называется *абстрактным*. Виртуальные функции в определении класса обозначаются следующим образом:

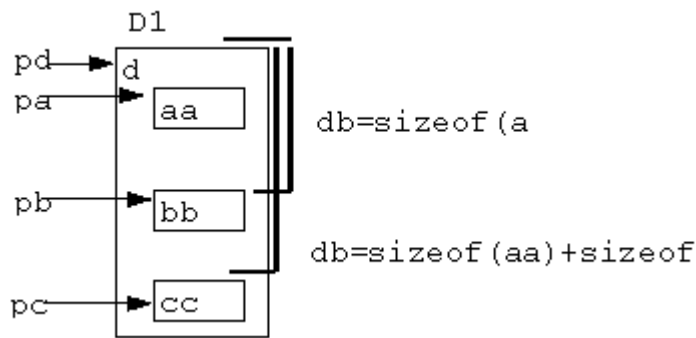
```
class base
{
public:
    virtual print()=0;
    virtual get()=0;
};
```

Определять тела этих функций не требуется.

Множественное наследование

Множественным наследованием называется процесс создания производного класса из двух и более базовых. В этом случае производный класс наследует данные и функции всех своих базовых предшественников. Существенным для реализации множественного наследования является то, что адреса объектов второго и последующих базовых классов не совпадают с адресом объекта производного класса. Этот факт должен учитываться транслятором при преобразовании указателя на производный класс в указатель на базовый и наоборот:

```
class d : public a, public b, public c { };
d D1;
pd = &D1;    // #define db sizeof(a)
pa = pd;     // #define dc sizeof(a)+sizeof(b)
pb = pd;     // pb = (char*)pd + db
pc = pd;     // pc = (char*)pd + dc
```



Такое действие выполняется компилятором как явно при преобразовании в программе типов указателей, так и неявно, когда в объекте производного класса наследуется функция из второго и последующих базовых классов. Для вышеуказанного примера при определении в классе bb функции f() и ее наследовании в классе "d" вызов D1.f() будет реализован следующим образом:

```
this = &D1;           // Указатель на объект производного класса
this = (char*)this + db // Смещение к объекту базового класса
b::f(this);           // Вызов функции в базовом классе
```

Механизм виртуальных функций при множественном наследовании имеет свои особенности. Во-первых, на каждый базовый класс в производном классе создается свой массив виртуальных функций (в нашем случае - для aa в d, для bb в d и для cc в d). Во-вторых, если функция базового класса переопределена в производном, то при ее вызове требуется преобразовать указатель на объект базового класса в указатель на объект производного. Для этого транслятор включает соответствующий код, корректирующий значение this в виде "заплаты", передающей управление командой перехода к переопределяемой функции, либо создает отдельные таблицы смещений.

Файловые потоки. Классы файловых потоков:

```
ifstream - файл ввода, производный от istream
ofstream - файл вывода, производный от ostream
fstream - файл ввода-вывода, производный от iostream
Флаги режимов работы с файлом:
enum ios::open_mode
{
in = 0x01,      // Открыть файл только для чтения
out = 0x02,     // Открыть файл только для записи
ate = 0x04,     // При открытии позиционироваться в конец файла
app = 0x08,     // Открыть существующий для дополнения
trunc = 0x10,   // Создание нового файла взамен существующего
nocreate=0x20,  // Не создавать новый файл при его отсутствии
noreplace=0x40, // Не создавать новый файл, если он существует
binary= 0x80 // Двоичный файл ("прозрачный" ввод-вывод без
// преобразования символов конца строки)
};
```

Конструкторы объектов (для классов `ifstream`, `ofstream`, `fstream`) и функции открытия/закрытия файлов:

```
ifstream(); // Без открытия файлов
ifstream( // С открытием файла в заданном
char *name, // режиме imode
int imode=ios::in,
int prot=filebuf::openprot);
ifstream(int fd); // С присоединением файла с дескрип-
// тором fd
ifstream( // То же, с явно заданным буфером
int fd,
char *buf, int sz);
void ifstream::open(
char *name, // Открытие файла в заданном режиме
int imode=ios::in,
int prot=filebuf::openprot);
void close(); // Закрыть файл
void setbuf(
char *p, int sz); // Установить буфер потока
int fd(); // Дескриптор открытого в потоке файла
int is_rtl_open(); // 1 - файл открыт в потоке
```

Унаследованные переопределения операторов позволяют проверять наличие ошибок в потоках в виде:

```
fstream ss;
if (ss) ... или if (!ss) ...
```

Обработка исключительных ситуаций

Средства обработки ошибочных ситуаций позволяют передать обработку исключений из кода, в котором возникло исключение, некоторому другому программному блоку, который выполнит в данном случае некоторые определенные действия. Таким образом, основная идея данного механизма состоит в том, что функция проекта, которая обнаружила непредвиденную ошибочную ситуацию, которую она не знает, как решить, генерирует сообщение об этом (бросок исключения). А система вызывает, по этому сообщению, программный модуль, который перехватит исключение и отреагирует на возникшее нештатное событие. Такой программный модуль называют «обработчик» или перехватчик исключительных ситуаций. И в случае возникновения исключения в его обработчик передаётся произвольное количество информации с контролем ее типа. Эта информация и является характеристикой возникшей нештатной ситуации.

Обработка исключений в C++ это обработка с завершением. Это означает, что исключается невозможность возобновления выполнения программы в точке возникновения исключения.

Для обеспечения работы такого механизма были введены следующие ключевые слова:

```
try - проба испытания;
catch - перехватить (обработать);
throw - бросать.
```

Кратко рассмотрим их назначение.

`try` - открывает блок кода, в котором может произойти ошибка; это обычный составной оператор:

```
try
{
```

```
код  
};
```

Код содержит набор операций и операторов, который и будет контролироваться на возникновение ошибки. В него могут входить вызовы функции пользователя, которые компилятор так же возьмет на контроль. Среди данного набора операторов и операций обязательно указывают операцию броска исключения: `throw`.

Операция броска `throw` имеет следующий формат:

`throw выражение;`

где - «выражение» определяет тип информации, которая и описывает исключение (например, конкретные типы данных).

`catch` - сам обработчик исключения, который перехватывает информацию:

```
catch ( тип параметр )  
{  
код  
}
```

Через параметр обработчику передаются данные определенного типа, описывающие обрабатываемое исключение.

Код определяет те действия, которые надо выполнить при возникновении данной конкретной ситуации. В C++ используют несколько форм обработчиков. Такой обработчик получил название параметризованный специализированный перехватчик

Перехватчик должен следовать сразу же после блока контроля, т.е. между обработчиком и блоком контроля не должно быть ни одного оператора. При этом в одном блоке контроля можно вызывать исключения разных типов для разных ситуаций, поэтому обработчиков может быть несколько.

В этом случае их необходимо расположить сразу же за контролирующим блоком последовательно друг за другом.

Кроме того, запрещены переходы, как извне в обработчик, так и между обработчиками.

Можно воспользоваться универсальным или абсолютным обработчиком:

```
catch ( . . . )  
{  
код  
}
```

где (...) - означают способность данного перехватчика обрабатывать информацию любого типа. Такой обработчик располагают последним в пакете специализированных обработчиков. Тогда, если исключение не будет перехвачено специализированными обработчиками, то будет выполнен последний - универсальный.

В случае не возникновения исключения, набор обработчиков будет обойден, т.е. проигнорирован.

Если же исключение было брошено, при возникновении критической ситуации, то будет вызван конкретный перехватчик при совпадении его параметра с выражением в операторе броска, т.е. управление будет передано найденному обработчику. После выполнения кода вызванного обработчика, управление передается оператору, который расположенный за последним перехватчиком, или проект корректно завершает работу.

Существенное отличие вызова конкретного обработчика от вызова обычной функции заключается в следующем: при возникновении исключения и передаче управления определенному обработчику, система осуществляет вызов всех деструкторов для всех объектов классов, которые были созданы с момента начала контроля и до возникновения исключительной ситуации с целью их уничтожения.

Блоки `try`, как составные блоки могут быть вложены:

```
try {  
...  
try  
{  
...  
}  
...  
}
```

тогда, в случае возникновения исключения в некотором текущем блоке, поиск обработчика последовательно продолжается в блоках, предшествующих уровням вложенности с продолжением вызова деструкторов.

НЕКОТОРЫЕ КРИПТОГРАФИЧЕСКИЕ ЭЛЕМЕНТЫ

В предстоящей Вам работе необходимо будет использовать некоторые элементы криптографии, такие как шифрование, дешифрование, сессионный ключ.

В настоящее время используется большое количество программ и библиотек для работы с криптографическими элементами, например, свободно распространяемый продукт Crypto++. Наша цель стоит не в изучении алгоритмов и способов шифрования, а также их программирования, а способность программным образом использовать уже известные и относительно простые способы некоторой защиты информации.

Для примера будем рассматривать возможность работы с системой защиты Windows под названием CRYPTO API.

CryptoAPI— интерфейс программирования приложений, который обеспечивает разработчиков Windows-приложений стандартным набором функций для работы с криптопровайдером. Входит в состав операционных систем Microsoft. Большинство функций CryptoAPI поддерживается, начиная с Windows 2000. CryptoAPI поддерживает работу с асимметричными и симметричными ключами, то есть позволяет шифровать и расшифровывать данные, а также работать с электронными сертификатами. Набор поддерживаемых криптографических алгоритмов зависит от конкретного криптопровайдера.

ОСНОВНЫЕ ПОНЯТИЯ И ТЕРМИНЫ

Криптография – наука о защите данных. Алгоритмы криптографии с помощью математических методов комбинируют входной открытый текст и ключ шифрования, в результате чего получаются зашифрованные данные. Применение криптографии обеспечивает относительно надёжную передачу данных и предотвращение их получения несанкционированной стороной. Применяя хороший алгоритм шифрования, можно максимально усложнить взлом защиты и получения открытого текста подбором ключа.

Шифрование – защита информации от несанкционированного просмотра или использования, особенно при передаче по линиям связи.

Дешифрование – процесс, обратный процессу шифрованию.

Аутентификация (authentication) – проверка подлинности, этот процесс надежного определения подлинности поддерживающих связь компьютеров. Аутентификация основана на методах криптографии, и это гарантирует, что нападающий или прослушивающий сеть не сможет получить информацию, необходимую для рассекречивания пользователя или другого объекта. Аутентификация позволяет поддерживающему связь объекту доказать свое тождество другому объекту без пересылки незащищенных данных по сети. Без "сильной" (strong) аутентификации и поддержания целостности, данных любые данные и компьютер, который их послал, являются подозрительными.

Шифр – обеспечивает возможность передачи сообщения по незащищенным каналам (не обязательно сетевым) с защитой этого сообщения от прочтения посторонними лицами.

Цифровая подпись – это двоичные данные небольшого объема, обычно не более 256 байт. Цифровая подпись есть не что иное, как результат работы хеш-алгоритма над исходными данными, зашифрованный закрытым ключом отправителя. Проще говоря, берем исходные данные, получаем из них хеш и шифруем хеш своим закрытым ключом (с помощью асимметричного алгоритма).

Сертификат – средство, позволяющее гарантированно установить связь между переданным открытым ключом и передавшей его стороной, владеющей соответствующим личным ключом. Сертификат представляет собой набор данных, зашифрованных с помощью цифровой, или электронной, подписи. Информация сертификата подтверждает истинность открытого ключа и владельца соответствующего личного ключа.

В сфере защиты компьютерной информации криптография применяется в основном для: шифрования и дешифрования данных; а также создания и проверки цифровых

подписей. Шифрование данных позволяет ограничить доступ к конфиденциальной информации, сделать ее нечитаемой и непонятной для посторонних.

Применение цифровых подписей оставляет данные открытыми, но дает возможность верифицировать отправителя и проверять целостность полученных данных.

Для защиты информации специалистами Microsoft был разработан интерфейс CryptoAPI, который является основой средств защиты Microsoft Internet Security Framework. Он позволяет создавать приложения, использующие криптографические методы и обеспечивает базовые функции защиты для безопасных каналов и подписи кода.

Реализация CryptoAPI позволяет интегрировать со своими приложениями усиленные средства шифрования. Интерфейс CryptoAPI обеспечивает API высокого уровня для аутентификации, подписи, шифрования/дешифрации, а также полную инфраструктуру защиты с общим ключом.

Благодаря данной инфраструктуре, можно воспользоваться функциями управления сертификатами, такими как запрос на создание сертификата, его сохранение или верификация.

Программный интерфейс CryptoAPI фирмы Microsoft предоставляет возможности для добавления в приложение функций аутентификации, шифрования, дешифрования и электронной подписи. Задачами CryptoAPI являются:

- Аутентификация сетевых пользователей
- Шифрование и дешифрование сетевых сообщений
- Шифрование и дешифрование данных
- Создание цифровой подписи и ее подтверждение.

Открытость интерфейса: открытая архитектура CryptoAPI позволяет выбирать Cryptographic Service Provider (CSP) по своему усмотрению. CryptoAPI доступен в операционных системах Windows, Macintosh и UNIX. Кроме того, в CryptoAPI поддерживаются стандартные форматы сертификатов X.509 версии 3, ASN.1 и DER.

Криптопровайдеры, инициализация и деинициализация

Любой сеанс работы с CryptoAPI начинается с инициализации (получения контекста). Инициализация выполняется при помощи функции **CryptAcquireContext**. В качестве параметров эта функция принимает имя контейнера ключей, имя криптопровайдера, тип провайдера и флаги, определяющие тип и действия с контейнером ключей и режим работы криптопровайдера:

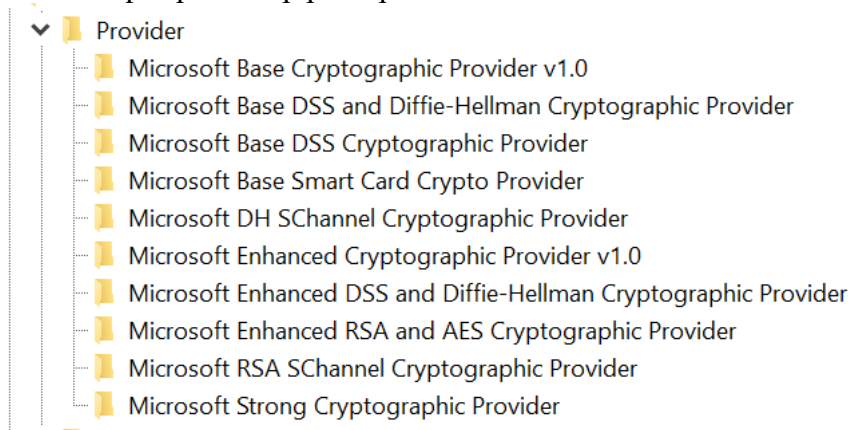
Криптопровайдер – это независимый модуль, содержащий библиотеку криптографических функций со стандартизованным интерфейсом. Криптопровайдер отвечает за реализацию функций интерфейса, а также играет роль хранилища для ключей всех типов. Подобная архитектура позволяет переходить от одного провайдера к другому с минимальными изменениями исходного кода, так как интерфейс (т. е. сами функции) не меняется.

Список криптопровайдеров Microsoft представлен в следующей таблице

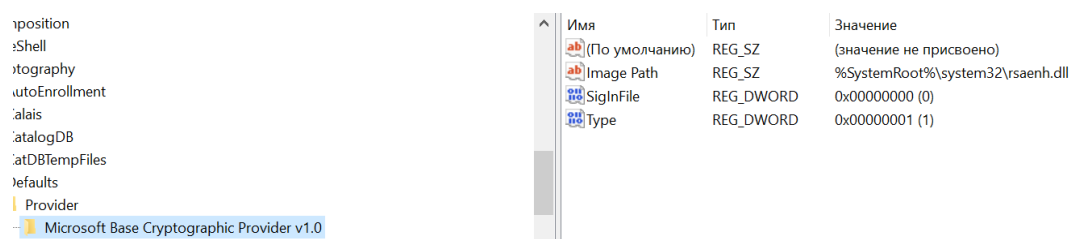
Провайдер	Описание
Microsoft Base Cryptographic Provider	Включает большой набор криптографических функций; может экспортироваться за пределы США
Microsoft Strong Cryptographic Provider	Расширяет Microsoft Base Cryptographic Provider ; доступен, начиная с версии Windows 2000
Microsoft Enhanced Cryptographic Provider	Представляет собой Microsoft Base Cryptographic Provider с более длинными ключами и дополнительными криптоалгоритмами.

Microsoft AES Cryptographic Provider	Microsoft <i>Enhanced Cryptographic</i> Provider с поддержкой <i>AES</i>
Microsoft DSS Cryptographic Provider	Предоставляет функции <i>хеширования</i> , генерации и <i>проверки ЭЦП</i> с использованием алгоритмов <i>Secure Hash Algorithm (SHA)</i> и <i>Digital Signature Standard (DSS)</i>
Microsoft Base DSS and Diffie-Hellman Cryptographic Provider	Помимо функциональности <i>DSS Cryptographic Provider</i> , поддерживает схему обмена ключами Диффи-Хеллмана
Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider	Поддерживает схему обмена ключами Диффи-Хеллмана, функции <i>хеширования</i> , генерации и <i>проверки ЭЦП</i> , соответствующие стандарту <i>FIPS 186-2</i> , <i>симметричное шифрование</i> на базе <i>RC4</i>
Microsoft DSS and Diffie-Hellman/SChannel Cryptographic Provider	Поддерживает хеширование и <i>ЭЦП</i> на базе <i>DSS</i> ; схему обмена ключами Диффи-Хеллмана, генерацию и экспорт этих ключей; получение ключей для протоколов <i>SSL3</i> и <i>TLS1</i>
Microsoft RSA/SChannel Cryptographic Provider	Поддерживает хеширование и <i>ЭЦП</i> , а также получение ключей для протоколов <i>SSL2</i> , <i>PCT1</i> , <i>SSL3</i> и <i>TLS1</i>
Microsoft RSA Signature Cryptographic Provider	Предоставляет функциональность для реализации <i>ЭЦП</i>

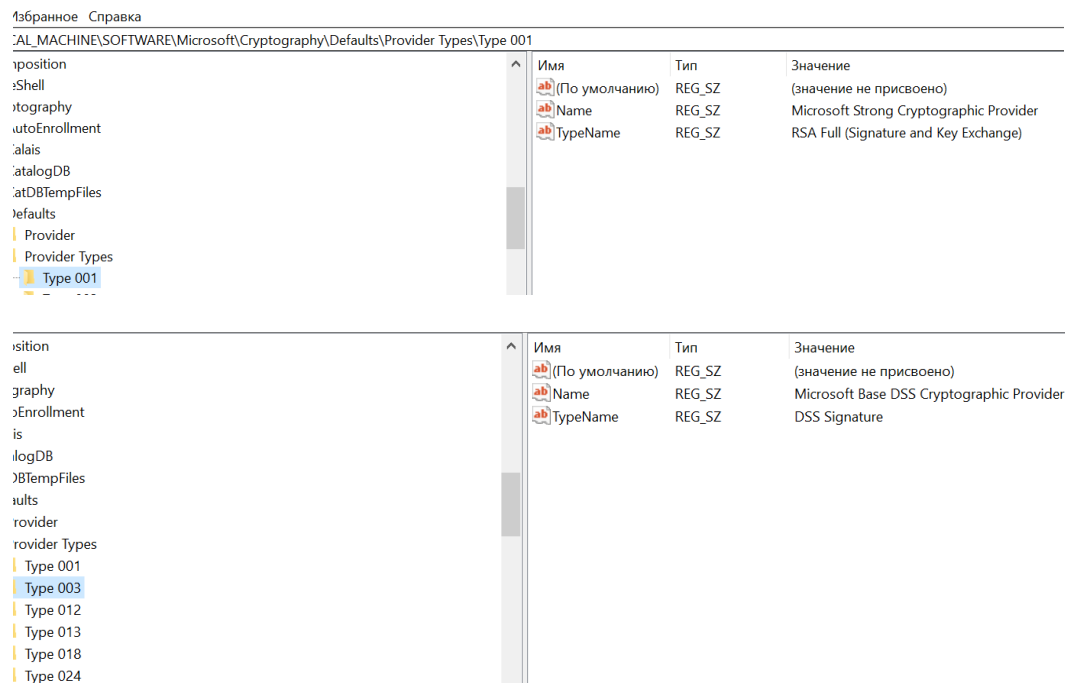
Со списком криптопровайдеров, представленных в Вашей системе, можно ознакомиться через редактор реестра



Там же можно увидеть тип провайдера (возможности)



и возможности соответствующего типа



Прежде чем использовать какие-либо функции Crypto API, необходимо запустить криптопровайдер. Делается это с помощью функции CryptAcquireContext:

```
BOOL CryptAcquireContext (
    HCRYPTPROV* hCryptProvider,           // дескриптор провайдера
    LPCTSTR pszContainer,                 // имя контейнера ключей
    LPCTSTR pszProvider,                 // имя провайдера
    DWORD dwProvType,                   // тип провайдера
    DWORD dwFlags                        // флаги
)
```

где:

hCryptProvider - указатель на дескриптор CSP, является выходным параметром. По завершении процесса работы с криптопровайдером(CSP), необходимо вызвать функцию CryptReleaseContext для освобождения(очистки) дескриптора и контейнера ключей.

pszContainer - имя ключевого контейнера. Это строка с нулевым завершением, которая идентифицирует контейнер ключей для CSP. Это имя не зависит от метода, используемого для хранения ключей(может быть массивом символов, строкой). Некоторые CSP хранят свои контейнеры ключей внутри (в аппаратном обеспечении), некоторые используют системный реестр, а другие-файловую систему. В большинстве случаев, когда *dwFlags* имеет значение CRYPT_VERIFYCONTEXT, *pszContainer* должен иметь значение NULL

pszProvider - имя провайдера. Это строка с нулевым завершением, которая идентифицирует имя используемого провайдера.

dwProvType – тип провайдера:

- PROV_RSA_FULL
- PROV_RSA_AES
- PROV_RSA_SIG
- PROV_RSA_SCHANNEL
- PROV_DSS
- PROV_DSS_DH
- PROV_DH_SCHANNEL
- PROV_FORTEZZA
- PROV_MS_EXCHANGE

- PROV_SSL

Более подробное описание типа криптопровайдера можно посмотреть в документации <https://docs.microsoft.com/ru-ru/windows/win32/seccrypto/cryptographic-provider-types>

dwFlags – флаги работы:

CRYPT_VERIFYCONTEXT, CRYPT_NEWKEYSET, CRYPT_MACHINE_KEYSET, CRYPT_DELETEKEYSET, CRYPT_SILENT, CRYPT_DEFAULT_CONTAINER_OPTIONAL.

Более подробное описание типа криптопровайдера можно посмотреть в документации <https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptacquirecontextw>

Кроме инициализации криптопровайдера данную функцию можно использовать для создания и удаления контейнеров ключей. Для этого параметру *dwFlags* присваивается значение, соответственно, CRYPT_NEWKEYSET и CRYPT_DELETEKEYSET. Функция *CryptAcquireContext* работает в два этапа: сначала она ищет криптопровайдер по имени и типу, указанному в аргументах, а затем контейнер ключей с заданным именем. По окончании работы с криптопровайдером необходимо вызвать функцию *CryptReleaseContext*.

Криптопровайдеры отличаются друг от друга:

- составом функций (например, некоторые криптопровайдеры не выполняют шифрование данных, ограничиваясь созданием и проверкой цифровых подписей);
- требованиями к оборудованию (специализированные криптопровайдеры могут требовать устройства для работы со смарт-картами для выполнения аутентификации пользователя);
- алгоритмами, осуществляющими базовые действия (создание ключей, хеширование и пр.).

По составу функций и обеспечивающих их алгоритмов криптопровайдеры подразделяются на типы. Например, любой CSP типа PROV_RSA_FULL поддерживает как шифрование, так и цифровые подписи, использует для обмена ключами и создания подписей алгоритм RSA, для шифрования — алгоритмы RC2 и RC4, а для хеширования — MD5 и SHA. В зависимости от версии операционной системы состав установленных криптопровайдеров может существенно изменяться. Однако на любом компьютере с Windows можно найти Microsoft Base Cryptographic Provider, относящийся к типу PROV_RSA_FULL. Именно с этим провайдером по умолчанию будут взаимодействовать все программы. Приложение обращается к криптопровайдеру не напрямую, а через *CryptoAPI*. Первым делом, хотя бы из любопытства, выясним, какие же криптопровайдеры установлены в системе.

Для определения имени криптопровайдера, его типа, реализации, версии, поддерживаемых алгоритмов в *CryptoAPI* используются следующие функции:

CryptEnumProviders (*i*, резерв, флаги, тип, имя, длина_имени) — возвращает имя и тип *i*-го по порядку криптопровайдера в системе (нумерация начинается с нуля);

CryptGetProvParam (провайдер, параметр, данные, размер_данных, флаги) — возвращает значение указанного параметра провайдера, например, версии (второй параметр при вызове функции — PP_VERSION), типа реализации (программный, аппаратный, смешанный — PP_IMPTYPE), поддерживаемых алгоритмов (PP_ENUMALGS). Список поддерживаемых алгоритмов при помощи этой функции может быть получен следующим образом: при одном вызове функции возвращается информация об одном алгоритме; при первом вызове функции следует передать значение флага CRYPT_FIRST, а при последующих флаг должен быть равен 0;

CryptReleaseContext (провайдер, флаги) — освобождает дескриптор криптопровайдера.

Каждая из этих функций, как и большинство других функций *CryptoAPI*, возвращает логическое значение, равное *true*, в случае успешного завершения, и *false* — если возникли ошибки. Код ошибки может быть получен при помощи функции *GetLastError*.

Шифрование

Базовая функция шифрования данных имеет следующее объявление:

```
BOOL CryptEncrypt(HCRYPTKEY hKey,  
                  HCRYPTHAS hHash,  
                  BOOL Final,  
                  DWORD dwFlags,  
                  BYTE* pbData,  
                  DWORD* pdwDataLen,  
                  DWORD dwBufLen);
```

Первым параметром данной функции передается хендл сессионного ключа, применяемого для шифрования. Второй параметр достаточно редко используется и предназначен для получения хеша данных одновременно с их шифрованием. Такая возможность достаточно полезна при формировании одновременно как шифрованных данных, так и цифровой подписи этих же данных.

Эта функция может обрабатывать данные блоками. То есть нет необходимости сразу загружать в память целиком весь массив данных, а лишь потом передавать ссылку на него криптографической функции. Достаточно передавать массив данных поблочно, специальным образом отметив лишь последний блок данных (это обычно нужно, чтобы криптопровайдер провел некоторые действия после использования сессионного ключа). Для указания того, что это последний блок данных, в функции **CryptEncrypt** используется третий параметр *Final*. Четвертый параметр служит указателем на массив входных/выходных данных. Здесь нужно сразу отметить некоторую общую схему работы с данными в Crypto API. Если возвращаемые данные могут быть любого размера (а это возможно, ведь, скажем, в алгоритме может происходить простая замена, когда одна буква кодируется четырьмя цифрами), то работа с функцией состоит из двух этапов. На первом этапе в функцию передается общий размер входных данных и NULL в качестве ссылки на сам массив выходных данных. Функция возвращает длину выходного массива данных, пользователь инициализирует память необходимого размера и лишь затем заново передает функции ссылку на этот массив. Такая же схема используется и в работе с функцией **CryptEncrypt**. Параметр *pdwDataLen* служит для возврата размера данных, возвращаемых функцией. Параметр *dwBufLen* служит для указания длины входного буфера данных. Параметр *dwFlags* обычно не используется и устанавливается в 0.

Расшифровывание

Базовая функция расшифровывания имеет следующее описание:

```
BOOL CryptDecrypt(HCRYPTKEY hKey,  
                  HCRYPTHASH hHash,  
                  BOOL Final,  
                  DWORD dwFlags,  
                  BYTE* pbData,  
                  DWORD* pdwDataLen);
```

Первым параметром данной функции передается инициализированный контекст сессионного ключа, применяемого для расшифровывания данных. Второй параметр, как и в предыдущем примере, связан, по большей части, с функцией получения и проверки цифровой подписи. Обычно он не используется и устанавливается в 0. Параметр *dwFlags* чаще всего не используется и также устанавливается в 0. Параметры *pbData* и *pdwDataLen* используются точно так же, как и у **CryptEncrypt** и представляют собой ссылку на входной/выходной массив данных и длину этого массива данных.

Криптографические ключи и выполняемые функции

В CryptoAPI существуют ключи двух типов:

сессионные ключи (session keys)

пары открытый/закрытый ключ (public/private key pairs).

Сессионные ключи - это симметричные ключи, так как один и тот же ключ применяется и для шифрования, и для расшифровки. Сессионные ключи меняются. Алгоритмы, использующие сессионные ключи (так называемые симметричные алгоритмы), – RC2, RC4, DES. Microsoft RSA Base Provider работает с 40-разрядными сессионными ключами.

Пары открытый/закрытый ключ используются в так называемых асимметричных алгоритмах шифрования. Если шифрование выполнялось одним ключом из пары, то дешифрование производится другим. Открытые (public) ключи могут передаваться другим лицам для проверки цифровых подписей и шифрования пересылаемых данных. Длина открытого ключа в Microsoft RSA Base Provider составляет 512 разрядов. Закрытые (private) ключи не могут быть экспортированы; они используются для создания цифровых подписей и дешифровки данных. Закрытый ключ должен быть известен только его владельцу. При шифровании с открытым ключом жизненно важна абсолютно достоверная ассоциация открытого ключа и передавшей его стороны, поскольку в обратном случае возможна подмена открытого ключа и осуществление несанкционированного доступа к передаваемым зашифрованным данным. Необходим механизм, гарантирующий достоверность корреспондента, например, применение сертификата, созданного авторизованным генератором сертификатов. Обычно сертификаты содержат дополнительную информацию, позволяющую идентифицировать владельца личного ключа, соответствующего данному открытому ключу. Сертификат должен быть подписан авторизованным генератором сертификатов. Выполняемые функции:

Функции кодирования сертификатов:

Эти функции управляют сертификатами и сопутствующими данными через сеть OSI (соединение открытых систем, семиуровневая модель) как описано в CCITT X.200. Методы OSI, описывающие абстрактные объекты, используют абстрактную синтаксическую нотацию один (ASN.1), как описано в CCITT X.209.

Функции базы сертификатов:

Используются для хранения сертификатов и управления ими. Пользователь со временем может собрать весьма много сертификатов. Обычно это сертификаты, описывающие самого пользователя и сертификаты сущностей с которыми он взаимодействует. Обычно для каждой сущности есть несколько сертификатов - связей, используемых для проверки отслеживания существующих сертификатов у авторитета сертификатов (обычно это сайт с сертификатами, отвечающий своим авторитетом за их верность).

Базовые криптографические функции:

Используются для наиболее полного использования криптографических возможностей в приложении. Это функции, взаимодействующие с провайдером.

Функции сообщений низкого уровня:

Используются для быстрого создания сообщений отвечающих требованиям PKCS#7(RFC2315 , RFC2316). Эти функции предназначены для шифрования данных при передаче и дешифровке их при приеме, а также для дешифровки и проверке подписей получаемых сообщений.

Упрощенные функции сообщений:

Они находятся на верхнем уровне функций сообщений и в принципе представляют низкоуровневые функции сообщений и сертификатов в одном. Они уменьшают число вызовов функций для приложения.

Хранение ключей

Криптопровайдер отвечает за хранение и разрушение ключей. Программист не имеет доступа непосредственно к двоичным данным ключа, за исключением операций экспорта

открытых ключей. Вся работа с ключами производится через дескрипторы (handle). В CryptoAPI ключи для шифрования / дешифрования и создания / проверки подписей разделены. Называются они соответственно «пара для обмена ключами» и «пара для подписи». База данных ключей состоит из контейнеров, в каждом из которых хранятся ключи, принадлежащие определенному пользователю. Контейнер ключей имеет уникальное имя и содержит пару для обмена и пару для подписи. Все ключи хранятся в защищенном виде. По умолчанию для каждого пользователя создается контейнер с именем этого пользователя.

Генерация ключей и обмен ключами

Для генерации ключей в CryptoAPI предусмотрены две функции –CryptGenKey и CryptDeriveKey. Первая из них генерирует ключи случайным образом, а вторая – на основе пользовательских данных. При этом гарантируется, что для одних и тех же входных данных CryptDeriveKey всегда выдает один и тот же результат. Это способ генерации ключей может быть полезен для создания симметричного ключа шифрования на базе пароля.

BOOL WINAPI CryptGenKey (HCRYPTPROV hProv, ALG_ID Algid, DWORD dwFlags, HCRYPTKEY* phKey);

Первый и четвертый параметры говорят сами за себя. Вторым параметром передается идентификатор алгоритма шифрования, для которого генерируется ключ (например, CALG_3DES). При генерации ключевых пар RSA для шифрования и подписи используются специальные значения AT_KEYEXCHANGE и AT_SIGNATURE. Третий параметр задает различные опции ключа, которые зависят от алгоритма и провайдера. Например, старшие 16 битов этого параметра могут задавать размер ключа для алгоритма RSA. Подробное описание всех флагов можно найти в MSDN.

Список алгоритмов(ALG_ID), реализующих шифрование с их числовыми константами:

CALG_MD2	= 32769;
CALG_MD4	= 32770;
CALG_MD5	= 32771;
CALG_SHA	= 32772;
CALG_SHA1	= 32772;
CALG_DES	= 26113;
CALG_3DES_112	= 26121;
CALG_3DES	= 26115;
CALG_RC2	= 26114;
CALG_RC4	= 26625;

Набор именованных констант - идентификаторов криптографических алгоритмов.

В зависимости от провайдера могут отличаться классом алгоритма, типом, размером, например для CALG_RC4:

Криптопровайдер	Класс алгоритма	тип	Размер (по умолчанию/ минимальный/ максимальный)
Microsoft Base Cryptographic Provider v1.0	шифрование	блочный	40/40/56
Microsoft Base DSS and Diffie-Hellman	шифрование	поточный	40/40/56

Cryptographic Provider			
Microsoft Base Smart Card Crypto Provider	шифрование	поточный	128/40/128
Microsoft DH Schannel Cryptographic Provider	шифрование	поточный	40/40/128

Идентификатор алгоритма	Класс алгоритма	Тип алгоритма	Комментарий
CALG_MD2	Хеширование	любой	MD2, 128 бит
CALG_MD4	Хеширование	любой	MD4, 128 бит
CALG_MD5	Хеширование	любой	MD5, бит
CALG_SHA, CALG_SHA1	Хеширование	любой	SHA-1, бит
CALG_DES	Симметричное шифрование	Блочный - 64 бита	DES с ключом 56 бит
CALG_3DES_112	Симметричное шифрование	Блочный - 64 бита	Тройной DES с ключом 112 бит
CALG_3DES	Симметричное шифрование	Блочный - 64 бита	Тройной DES с ключом 168 бит
CALG_RC2	Симметричное шифрование	Блочный - 64 бита	RC2 с переменной длиной ключа
CALG_RC4	Симметричное шифрование	Поточный	RC4 с переменной длиной ключа

Обмен ключами в CryptoAPI реализуется с помощью функций CryptExportKey и CryptImportKey

После окончания работы с ключом, его нужно уничтожить вызовом CryptDestroyKey. При этом закрытый ключ сохраняется в контейнере (если, конечно, не использовался режим CRYPT_VERIFYCONTEXT), а сессионные ключи уничтожаются совсем.

Пример: задана строка символов, осуществить ее шифрование, ее запись в файл, считывание из файла и ее последующую расшифровку. использовать для шифрования сессионный ключ.

Алгоритм программы следующий:

1. Подключение к провайдеру
2. Генерация ключа и его сохранение в памяти(можно файле)
3. Шифрование
4. Открытие файла
5. Запись зашифрованной строки в файл
6. Закрытие файла
7. Открытие файла
8. Считывание записи из файла
9. Расшифровка с помощью сохраненного ключа
10. Закрытие файла

Программа реализована на языке программирования C++ в среде Visual Studio 17

```

#include "pch.h"
#include <iostream>
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h>
using namespace std;
//using namespace Cryptograph

DWORD dwIndex = 0;
DWORD dwType;
DWORD cbName;
LPTSTR pszName,x;
void main() {

    HCRYPTPROV hProv;
    HCRYPTKEY hSessionKey;

    // Получение контекста криптопровайдера
    if (!CryptAcquireContext(&hProv, NULL, NULL,
        PROV_RSA_FULL, CRYPT_VERIFYCONTEXT))
    {
        std::cout << "CryptAcquireContext" << std::endl;
        return;
    }

    std::cout << "Cryptographic provider initialized" << std::endl;

    // Генерация сессионного ключа
    if (!CryptGenKey(hProv, CALG_RC4,
        CRYPT_EXPORTABLE, &hSessionKey))
    {
        std::cout << "CryptGenKey" << std::endl;
        return;
    }

    std::cout << "Session key generated" << std::endl;

    // Данные для шифрования
    char string[] = "TestTest";
    char string1[100];
    char string2[100];
    //memcpy(string2, '0', 100);
    strcpy_s(string1, string);
    DWORD count = strlen(string);

    // Шифрование данных
    if (!CryptEncrypt(hSessionKey, 0, true, 0, (BYTE*)string,
        &count, strlen(string)))
    {
        std::cout << "CryptEncrypt" << std::endl;
        return;
    }
}

```



```

}

std::cout << "Encryption completed" << std::endl;

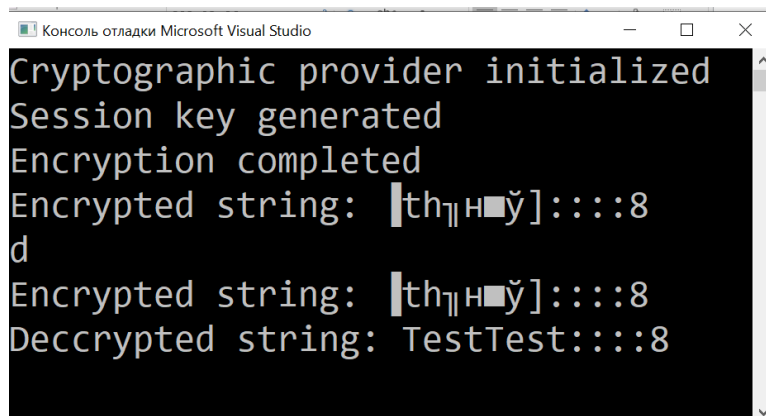
// Тестовый вывод на экран
std::cout << "Encrypted string: " << string << "::::"
    << strlen(string) << std::endl;

//-----
// работа с файлом
//-----
FILE *out = nullptr, *in = nullptr;
char c;
fopen_s(&out, "AAA.ddd", "w");
fwrite(string, strlen(string), 1, out);
fclose(out);
cin >> c;
fopen_s(&in, "AAA.ddd", "r");
fread(string1, strlen(string), 1, in);
fclose(in);

DWORD count1 = strlen(string1);
std::cout << "Encrypted string: " << string1 << "::::"
    << strlen(string1) << std::endl;

if (!CryptDecrypt(hSessionKey, 0, true, 0, (BYTE*)string1, &count1))
{
    std::cout << "CryptDecrypt" << std::endl;
    return;
}
std::cout << "Decrypted string: " << string1 << "::::"
    << strlen(string1) << std::endl;
}

```



```

Консоль отладки Microsoft Visual Studio
Cryptographic provider initialized
Session key generated
Encryption completed
Encrypted string: [thн[ ]]::8
d
Encrypted string: [thн[ ]]::8
Decrypted string: TestTest:::8

```

Приложение №1 Бланки задания на курсовую



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-2 «Прикладные информационные технологии»

КУРСОВОЙ ПРОЕКТ (РАБОТА)

по дисциплине: «Языки программирования»

(наименование дисциплины)

Тема курсового проекта (работы) _____

Студент группы _____
учебная группа, фамилия, имя, отчество студента

подпись студента

Руководитель курсового проекта (работы) _____
должность, звание, ученая степень, _____
подпись руководителя

Рецензент (при наличии) _____
должность, звание, ученая степень _____
подпись рецензента

Работа представлена к защите « ____ » _____ 20 __ г.

Допущен к защите « ____ » _____ 20 __

Список литературы

1. Род Хаггарти. Дискретная математика для программистов: 2-е издание, исправленное. М.: Техносфера 2018 — 400 с.
2. Новиков Ф. Дискретная математика: Учебник для вузов. 3-е изд. Стандарт третьего поколения. — СПб.: Питер, 2017. — 496 с.: ил. — (Серия «Учебник для вузов»).
3. Ерусалимский Я. М. Дискретная математика. Теория, задачи, приложения. — 2018. — 476 с.
4. Тишин В. В. Дискретная математика в примерах и задачах. — 2-е изд., испр. — СПб.: БХВ-Петербург, 2016. — 336 с.: (Учебная литература для вузов)
5. Авдошин С. М., Набсбин А. А. Дискретная математика. Модулярная алгебра, криптография, кодирование. — М.: ДМК Пресс, 2017. - 352 с.: ил.
6. Дискретная математика: прикладные задачи и сложность алгоритмов : учебник и практикум для академического бакалавриата / А. Е. Андреев, А. А. Болотов, К. В. Коляда, А. Б. Фролов. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2018. — 317 с.
7. Задачи по дискретной математике / С. В. Борзунов, С. Д. Кургалин. — СПб.: БХВ-Петербург, 2016. — 528 с.: ил. — (Учебная литература для вузов)
8. Судоплатов С. В. Дискретная математика : учебник и практикум для академического бакалавриата / С. В. Судоплатов, Е. В. Овчинникова. — 5-е изд., испр. и доп. — М. : Издательство Юрайт, 2018. — 279 с.
9. Гисин, В. Б. Дискретная математика : учебник и практикум для академического бакалавриата / В. Б. Гисин. — М. : Издательство Юрайт, 2018. — 383 с.
10. Брайан У. Керниган, Роб Пайк, Практика программирования М.: Вильямс 2017. 288с.
11. Роберт Мартин. Чистая архитектура. Искусство разработки программного обеспечения СПб.: Питер 2018 352с
12. Майкл Ховард, Дэвид Лебланк, Джон Виега, Как написать безопасный код на C++, Java, Perl, PHP, ASP.NET М.: ДМК Пресс 2017 288с.
13. Сергей Никифоров, Прикладное программирование М.: Лань 2018 124с.
14. Э Фримен, Э Фримен, К Сьерра, Б Бейтс. Паттерны проектирования //СПб.: Питер. — 2018. 656 с.
15. И.Г. Гниденко, Ф.Ф. Павлов, Д.Ю. Федоров Технологии и методы программирования. М.: Юрайт 2017 235с.
16. Адитья Бхаргава, Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. М.: Издательство «Питер» 2017. 288 с.
17. Дино Эспозито, Андреа Сальтарелло, Microsoft .NET. Архитектура корпоративных приложений М.: Вильямс 2017 432с.